Safe Transfer Learning for Robotic Environments

Master's Thesis of

Florian Krone

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf Reussner
Second reviewer:	Prof. Dr. J. Marius Zöllner
Advisor:	M.Sc. Karam Daaboul
Second advisor:	M.Sc. Jakob Weinland

23. June 2022 – 23. December 2022

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 12. December 2022

. . . . . .

(Florian Krone)

Abstract

Deep reinforcement learning has shown impressive results when using complex features as observations. In recent years, the performance of agents using high dimensional image observations was improved significantly. However, it still lacks behind for complex tasks. In this work, we explore a novel approach to guide the exploration of an agent. We show that this can be utilized to significantly improve the performance of agents trained on high dimensional image observations. We call our approach the Preference Reward algorithm. Our approach uses reward shaping and can therefore be used with any reinforcement learning algorithm. The second major problem we focused on in this work is safe reinforcement learning. In robotic environments, the agent must be constrained to not reach certain states that for example could hurt humans or the robot itself. We propose two new algorithms to train agents with such constraints in mind. The algorithms are called Safety Training and Safety Evaluation. Our algorithms are designed around actor critic methods. Specifically, we evaluated them with a Soft Actor-Critic (SAC).

Zusammenfassung

Mit Deep Reinforcement Learning sind beeindruckende Ergebnisse möglich, wenn komplexe Features als Observationen verwendet werden. In den letzten Jahren wurde die Leistung von Agenten, die hochdimensionale Bildobservationen verwenden deutlich verbessert. Bei komplexen Aufgaben liegt sie jedoch noch immer deutlich unter derer von Agenten mit komplexen Fetarures als Observationen. In dieser Arbeit untersuchen wir einen neuartigen Ansatz, um die Erkundung eines Agenten zu steuern. Wir zeigen, dass dieser Ansatz die Leistung von Agenten, die auf hochdimensionalen Bildobservationen trainiert werden, deutlich verbessern kann. Wir nennen unseren Ansatz den Preference Reward Algorithmus. Unser Ansatz verwendet Reward-Shaping und kann daher mit jedem Reinforcement Learning Algorithmus verwendet werden. Das zweite große Problem, auf das wir uns in dieser Arbeit konzentrieren haben, ist safe Reinforcement Learning. In Roboterumgebungen darf der Agent bestimmte bestimmte Zustände nicht erreichen, die zum Beispiel Menschen oder den Roboter selbst verletzen könnten. Wir schlagen zwei neue Algorithmen vor, um Agenten unter Berücksichtigung solcher Einschränkungen zu trainieren. Wir nennen die Algorithmen Safety Training und Safety Evaluation. Unsere Algorithmen sind auf der Grundlage von Actor Critic Methoden entwickelt. Insbesondere haben wir sie mit einer Soft Actor-Critic (SAC) evaluiert.

Contents

List of Figures

List of Tables

1 Introduction

Learning through interaction, this is the paradigm that is usually used to describe reinforcement learning (RL). An agent interacts with an environment and receives a reward signal for its actions. The agent then learns to maximize the reward signal. This basic idea behind reinforcement learning is derived from the way humans learn. [20] At least, this is what is often used to motivate reinforcement learning. However, this explanation forgets an essential part of the learning process. A child does not just start to speak because it taught itself to do so but imitates its parents. Thus the parents act as a teacher for the child. For more complex tasks, we actually employ teachers. Be it in school, or as driving instructors, for example. We do not just learn to drive a car but someone tells us how to do so. Although we might be able to learn how to drive purely from experience, it would be a massive security risk.

With this in mind, reinforcement learning clearly lacks the notion of a teacher. We expect an RL agent to learn complex tasks purely through interaction. In this work, we explore the possibility of extending the reinforcement learning framework with a teacher. The teacher is hidden behind a simple interface. This allows us to develop our approach agnostic to the actual teacher used in practice.

Obviously, a teacher requires knowledge about the task and how to solve it. A possible, yet very simple, the teacher can be designed around a human as a source of knowledge. However, RL algorithms tend to require a lot of samples to learn complex tasks, making this approach very tedious. Another possibility is to utilize another control algorithm, like trajectory optimization, as a teacher. Even other RL agents can be used as teachers.

One might question the need to train an RL agent using a teacher, if we already have a teacher at hand that can solve the same task. The motivation to train the RL agent is manifold. The teacher may only be able to solve a sub-task of the task the RL agent is trained on. The teacher can then still be used to guide the exploration of the RL agent. Another option is to use a different observation for the teacher and the student. If the student uses only images as observation, only a camera is needed as a sensor when using the agent in practice. To help the training, the teacher may use the information from additional sensors. This makes it cheaper to use the student in practice than the teacher. But even if the student and the teacher use the same observations to solve the same task, we benefit from having an RL agent. With modern RL algorithms, we get a stochastic policy that can adapt to changes in the environment. Also, the continued interaction with the environment when using the student in practice can be used to train the agent without a teacher further.

Typically, reinforcement learning aims to solve a task by maximizing a reward function. Additionally, a second goal arises when applying reinforcement learning to real-world problems, especially robotic tasks like those considered in this work. The agent needs to interact with its environment safely. A very obvious safety risk lies in the interaction with

humans for a robotic control task. The robot should never harm a human. The second contribution of this work is the development of two novel algorithms to train an agent so that it can be safely used in real-world scenarios.

The remainder of this work is structured as follows. In chapter 2 we establish the background needed for this work. This includes reinforcement learning in general and the specific algorithms used in his work. Following the preliminaries, we give an overview of related works in chapter 3. Chapter 4 first introduces the aforementioned teacher in the preference reward algorithm. Afterward, we propose two novel algorithms for safe RL, safety training, and safety evaluation. All algorithms are evaluated in chapter 5 with many experiments on three robotic control environments. We conclude in chapter 6 with a summary of the approaches and the results. Additionally, we give an outlook on possible future steps.

2 Preliminaries

2.1 Reinforcement Learning

In reinforcement learning (RL) an agent that interacts with an environment is trained to maximize a reward function. The agent takes actions based on observations from the environment. The problem is often formulated as a Markov decision process (MDP). The notation in the following sections is taken from OpenAI Spinning Up [24].

2.1.1 Markov Decision Process

Definition 2.1 *Given a*

- *Set of states* S
- *Set of actions* A
- *Transition probability function* $\mathrm{P} : S \times A \times S \to [0, 1]$
- *Reward function* $\mathrm{R} : S \times A \times S \to [0, 1]$
- *Start state distribution* ρ_0
- *Discount factor* γ

the tuple $(S, A, P, R, \rho_0, \gamma)$ *is called a Markov decision process (MDP).*

A trajectory τ is a sequence of states and actions $\tau = (s_0, a_0, s_1, a_1, \dots)$. At a discrete time step t, the transition function $P(s'|s, a) = P(s_{t+1} = s'|s_t = s, a_t = s)$ is given by the probability of reaching state s_{t+1} if the current state is s_t and action a_t is taken. According to the Markov property the next state s_{t+1} only depends on the current state s_t and action a_t.

The reward function $r_t = R(s_t, a_t, s_{t+1})$ describes the reward for a state-action-next-state triplet. R also denotes the return of a trajectory, given by the discounted sum of rewards:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.1}$$

The discount factor γ describes how import rewards in the future are in contrast to immediate rewards.

Most of the agents in this work do not use the states of the environment as observations, but an image of the environment. In this case the MDP is considered to be partially observable, as not all state information might be observable at every time step.

2.1.1.1 Constrained Markov Decision Process

Multiple definitions of constrained Markov decision processes (CMDP) exist. In this work a CMDP is a MDP extended with a cost function $C : S \times A \times S \to [0, 1]$ and a threshold $\delta \in \mathbb{R}_{\geq 0}$. Similar to the reward in equation 2.1, this can be extended for a trajectory:

$$C(\tau) = \sum_{t=0}^{\infty} \gamma^t c_t \tag{2.2}$$

with $c_t = C(s_t, a_t, c_{t+1})$ denoting the cost of a state-action-next-state triplet in a trajectory $\tau = (s_0, a_0, s_1, a_1, \dots)$. If $C(\tau) \leq \delta$ the trajectory τ fulfills the constraint.

2.1.2 The Reinforcement Learning Goal

The agent selects which action to take in the environment based on a policy π. RL aims to find the policy that maximizes the expected reward. Only stochastic policies $\pi(a|s)$ are used in this work. For a state s_t, an action a_t can be sampled from the policy: $a_t \sim \pi(\cdot|s_t)$. In the following, if the state is evident from the context, this will be written as $a \sim \pi$. When combined with the transition probability function, it is also possible to sample whole trajectories $\tau \sim \pi$.

Given the transition probability function P, the reward function R, and a policy π, the probability of a trajectory τ of length T can be written as:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.3}$$

The expected return $J_R(\pi)$ when using the policy π can then be written as:

$$J_R(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)] \tag{2.4}$$

The goal of RL, finding the optimal policy π^* that maximizes the expected reward, can then be denoted as:

$$\pi^* = \arg\max_{\pi} J_R(\pi) \tag{2.5}$$

Combining equations 2.1, 2.4, and 2.5 yields:

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\Big] \tag{2.6}$$

For a CMDP the expected cost when using policy π can be written as:

$$J_C(\pi) = \int_{\tau} P(\tau|\pi)C(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi}[C(\tau)] \tag{2.7}$$

This changes the RL goal for a CMDP to:

$$\pi^* = \arg\max_{\pi} J_R(\pi), \ s.t. \ J_C(\pi) \leq \delta \tag{2.8}$$

2.1.3 Value Functions

The value of a state s or a state-action pair (s, a) under policy π can be represented as the expected return when starting in s and following policy π. In the case of a state-action pair, the initial action is specified as a:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{2.9}$$

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{2.10}$$

If all actions are chosen by the optimal policy, the value functions are denoted in the following way:

$$V^{*}(s) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{2.11}$$

$$Q^{*}(s, a) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{2.12}$$

Based on the value functions, the Bellman equations can be formulated. They reformulate the value functions as the sum of the reward we expect to get for being in a state s and the expected return from the next state s', discounted by γ.

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\substack{a\sim\pi \\ s'\sim P(\cdot|s,a)}}[R(s, a, s') + \gamma V^{\pi}(s')] \tag{2.13}$$

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)}[R(s, a, s') + \gamma \mathop{\mathbb{E}}_{a'\sim\pi}[Q^{\pi}(s', a')]] \tag{2.14}$$

Again, when following the optimal policies, this changes to:

$$V^{*}(s) = \max_{a} \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)}[R(s, a, s') + \gamma V^{*}(s')] \tag{2.15}$$

$$Q^{*}(s, a) = \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)}[R(s, a, s) + \gamma \max_{a'}[Q^{*}(s', a')]] \tag{2.16}$$

2.1.4 Soft Actor-Critic

Soft Actor-Critic (SAC) [10] is one of the current state of the art RL algorithms. SAC uses entropy regularization to prevent the policy from only learning a single solution to the problem.

Definition 2.2 *Given a random variable X, distributed according to P, the* entropy *is given by:*

$$H(P) = \mathop{\mathbb{E}}_{X\sim P}[-\log P(X)] \tag{2.17}$$

The entropy of the policy is used to change the definition of the optimal policy from equation 2.6, as well as the value functions and bellman equations, to a trade-off between maximizing the return and the entropy of the policy. A parameter α is added to determine the importance of the entropy.

$$\pi^{*} = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[\sum_{t=0}^{\infty} \gamma^{t}\Big(R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t))\Big)\Big] \tag{2.18}$$

Therefore, also the value functions change:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H(\pi(\cdot|s_t))\Big)\Big|s_0 = s\Big] \tag{2.19}$$

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H(\pi(\cdot|s_t))\Big|s_0 = s, a_0 = a\Big] \tag{2.20}$$

As well as the Bellman equations. However, the state bellman equation is not needed for SAC. It is therefore omitted here.

$$\begin{aligned} Q^{\pi}(s,a) &= \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)}\Big[R(s,a,s') + \gamma \mathop{\mathbb{E}}_{a'\sim\pi}\Big[Q^{\pi}(s',a') + \alpha H(\pi(\cdot|s'))\Big]\Big] && (2.21)\\ &= \mathop{\mathbb{E}}_{\substack{s'\sim P(\cdot|s,a)\\ a'\sim\pi}}\Big[R(s,a,s') + \gamma\Big(Q^{\pi}(s',a') + \alpha H(\pi(\cdot|s'))\Big)\Big] && (2.22)\\ &= \mathop{\mathbb{E}}_{\substack{s'\sim P(\cdot|s,a)\\ a'\sim\pi}}\Big[R(s,a,s') + \gamma\Big(Q^{\pi}(s',a') - \alpha\log\pi(a'|s')\Big)\Big] && (2.23) \end{aligned}$$

In newer implementations of SAC, α is often learned during training. [9]

SAC is an off-policy method. This means that when executing an action in the environment, the resulting quintuple *state, action, reward, next state, done* (s, a, r, s', d) is not used for training directly, but stored in a replay buffer D. The flag *done* identifies, whether the episode was terminated after this action.

A SAC implementation consists of an actor, i.e. the policy π_θ and a critic, which in turn consists of two Q-functions Q_{ϕ_i} $i \in \{1,2\}$. The double-Q trick helps to make the training more robust. All three components are separate neural networks. θ and ϕ_i denote the parameters of the networks. For a training step, a batch of (s, a, r, s', d) transitions is sampled from the replay buffer.

2.1.4.1 Training the Critic

Since the Q-function as denoted in equation 2.23 is an expectation, it can be approximated using samples. For a given sample (s, a, r, s', d) from the replay buffer D, only the next action a' needs to be sampled from $\pi(\cdot|s')$:

$$Q^{\pi}(s,a) \approx r + \gamma(Q^{\pi}(s',a') - \alpha\log\ \pi(a'|s')),\ a' \sim \pi(\cdot|s') \tag{2.24}$$

In practice, this is used to create a target to train the Q-functions:

$$y(r,s',d) = r + \gamma(1-d)\Big(\min_{j=1,2} Q_{\overline{\phi}_i}(s',a') - \alpha\log\ \pi_\theta(a'|s')\Big),\ a' \sim \pi_\theta(\cdot|s') \tag{2.25}$$

The recursion in the Bellman equation utilizes two Q-functions parameterized with $\overline{\phi}_i$. These are called the target networks and represent older versions of the two Q-functions whose parameters are updated every few training steps with the new parameters ϕ_i. The use of target networks stabilizes the training. The policy π_θ is used to sample the following

action a'. The term $1 - d$ is necessary since the episode terminated in case of $d = 1$, and there is no return from future actions.

With all this in place, the two Q-functions can be trained using gradient descent with:

$$J_Q(\phi)_i = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right] \tag{2.26}$$

2.1.4.2 Training the Actor

The goal of SAC is to train a policy that is proportional to the exponential of the Q-function. This results in a stochastic policy that, with a high probability, proposes actions whose Q value is also high for a given state.

Definition 2.3 *For two probability distributions* p, q *on the same probability space X, the Kullback-Leibler divergence (KL) [13] is given by:*

$$D_{KL}(p||q) = \sum_{x\in X} p(x)\log\frac{p(x)}{q(x)} \tag{2.27}$$

The KL is a similarity measure between the two distributions. It is always non-negative and zero if and only if p = q. Nevertheless, it is not symmetric.

The policy parameters can be optimized by minimizing the KL of the policy and the exponential of the Q-function:

$$\mathop{\mathbb{E}}_{s\sim D}\left[D_{KL}\left(\pi_\theta(\cdot|s)\middle\|\frac{\exp\left(\frac{1}{\alpha}Q_\phi(s,\cdot)\right)}{Z_\phi(s)}\right)\right] \tag{2.28}$$

$$= \mathop{\mathbb{E}}_{s\sim D}\left[\sum_{a\in A}\pi_\theta(a|s)\log\left(\frac{\pi_\theta(a|s)Z_\phi(s)}{\exp\left(\frac{1}{\alpha}Q_\phi(s,a)\right)}\right)\right] \tag{2.29}$$

$$= \mathop{\mathbb{E}}_{\substack{s\sim D\\ a\sim\pi_\theta(\cdot|s)}}\left[\log\pi_\theta(a|s) + \log Z_\phi(s) - \frac{1}{\alpha}Q_\phi(s,a)\right] \tag{2.30}$$

The function $Z_\phi(s)$ is needed to normalize $\exp Q_\phi(s,\cdot)$, since the KL is only defined for two distributions. It was shown that $Z_\phi(s)$ is not present in the gradient with respect to θ. Therefore it is omitted in the future. Additionally the equation is multiplied by α to get the training objective:

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{\substack{s\sim D\\ a\sim\pi_\theta(\cdot|s)}}\left[\alpha\log\pi_\theta(a|s) - Q_\phi(s,a)\right] \tag{2.31}$$

This objective cannot be directly approximated through Monte Carlo estimation, because the the distribution that has to be sampled from depends on the parameters that are to be optimized. However, since the policy is supposed to be a Gaussian, the reparametrization trick can be used. Instead of having a probabilistic neural network for the policy that directly gives an action for each state, a double headed, deterministic network is used. The

two heads are the functions $\mu_\theta(s)$ and $\sigma_\theta(s)$. The distribution of actions for each state s is then given by:

$$f_\theta(s, \xi) = \mu_\theta(s) + \sigma_\theta(s)\xi, \ \xi \sim \mathcal{N}(0, I) \tag{2.32}$$

With this reparametrization, equation 2.31 can be rewritten as:

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{\substack{s\sim D \\ \xi\sim\mathcal{N}(0,I)}} \Big[\alpha \log \pi_\theta(f_\theta(s, \xi)|s) - Q_\phi(s, f_\theta(s, \xi))\Big] \tag{2.33}$$

Now, the distribution the expectation is taken over no longer depends on θ. Finally, as described in section 2.1.4.1, SAC uses two Q-functions to stabilize the training. Therefore, the more pessimistic of the two should be used in the training objective:

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{\substack{s\sim D \\ \xi\sim\mathcal{N}(0,I)}} \Big[\alpha \log \pi_\theta(f_\theta(s, \xi)|s) - \min_{i=1,2} Q_{\phi_i}(s, f_\theta(s, \xi))\Big] \tag{2.34}$$

With batches of states drawn from the replay buffer, this objective function is used to optimize the policy using gradient descent.

In practice, the reparametrization trick is extended with a tanh to ensure that the actions are in a valid range:

$$f_\theta(s, \xi) = \tanh(\mu_\theta(s) + \sigma_\theta(s)\xi), \ \xi \sim \mathcal{N}(0, I) \tag{2.35}$$

2.1.4.3 SAC algorithm

The SAC algorithm, outlined in algorithm 1, combines the two training objectives. First, the target networks are set to the same parameters as the initial Q-functions. Also the empty replay buffer is initialized and the initial state s is observed from the environment. Afterwards, the training loop starts. First, an action a is sampled from the policy for the current state. This action is executed in the environment returning the reward r, next state s' and an indicator if the episode is done d. The Quintuple (s, a, r, s', d) is added to the replay buffer. If the episode is done, the environment is reset and the initial state is loaded again. Afterwards, a batch is sampled from the replay buffer and a gradient descent update is done. First, the Q-function targets are calculated according to equation 2.25. The parameters for the two Q-functions and the policy are updated with the objectives from the equations 2.26 and 2.34. Lastly the weights of the target networks are updated.

Variations of this algorithm exist and include mainly different execution frequencies. It is possible to execute multiple environment steps without having a gradient descent in between, or do multiple updates after each environment step. Also the target networks may not be updated after ever step. The algorithm as described here, is the variant that was used during this work.

2.2 High Dimensional Observations

When talking about high dimensional observations in RL, this usually means images of the environment. This is in contrast the actual state which consists of vectors with the

Algorithm 1: Soft Actor-Critic

Input: entropy coefficient α, environment e, initial parameters θ, ϕ_1, ϕ_2, learning rates η_π, η_Q, target update rate ρ
Data: replay buffer D
Output: θ, ϕ_1, ϕ_2

// Initialize target networks
$\overline{\phi}_1 \leftarrow \phi_1, \overline{\phi}_2 \leftarrow \phi_2$
$s \leftarrow e.\text{INITIALSTATE}()$
for *each iteration* **do**
  // Interact with the environment
  $a \sim \pi_\theta(\cdot|s)$
  $r, s', d \leftarrow e.\text{STEP}(a)$
  $D.\text{APPEND}((s, a, r, s', d))$
  if $d = done$ **then**
    $e.\text{RESET}()$
    $s \leftarrow e.\text{INITIALSTATE}()$
  Sample a batch B of transitions from D
  // Compute Q-function targets
  foreach $(s, a, r, s', d) \in B$ **do**
    $y(r, s', d) \leftarrow r + \gamma(1-d)\left(\min_{j=1,2} Q_{\overline{\phi}_i}(s', a') - \alpha \log \pi_\theta(a'|s')\right),\ a' \sim \pi_\theta(\cdot|s')$
  // Update Q-functions and policy by one step of gradient descent
  for $i = 1, 2$ **do**
    $\phi_i \leftarrow \phi_i - \eta_Q \nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left[\left(Q_{\phi_i}(s, a) - y(r, s', d)^2\right)\right]$
  for $i = 1, 2, \ldots, |B|$ **do**
    $\xi_i \leftarrow \mathcal{N}(0, I)$
  $\theta \leftarrow \theta - \eta_\pi \nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left[\alpha \log \pi_\theta(f_\theta(s, \xi_i)|s) - \min_{i=1,2} Q_{\phi_i}(s, f_\theta(s, \xi_i))\right]$
  // Update target networks
  for $i = 1, 2$ **do**
    $\overline{\phi}_i \leftarrow \rho\overline{\phi}_i + (1-\rho)\phi_i$
  return θ, ϕ_1, ϕ_2

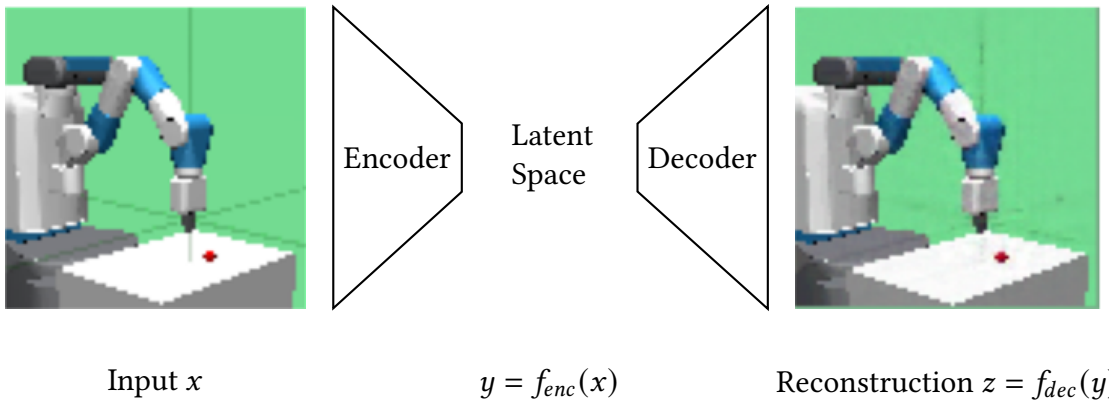


Figure 2.1: High level view of an autoencoder. The reconstructed image is the actual output from an autoencoder that was trained during this work for the input image.

position and velocity of the objects. Since it is not possible to extract the velocity from a single image, it is common practice to use multiple consecutive images as observations [14, 12, 18, 26]. This is called a frame stack.

In the first step, the images are converted to a latent space using an encoder. When using SAC, the latent space is then given to the actor and critic networks to produce actions and reward estimates. The only difference between using image and space observations lies in the encoder. State observations can be used directly with the actor and critic networks without the need for an encoder. The encoder consists of a multi layer CNN. The output of the last convolution is flattened to produce a one dimensional vector for the latent space.

Different approaches have been attempted to improve the performance of RL with high dimensional observations. They include auxiliary losses [26, 19, 18] as well as data augmentations [12]. In this work, autoencoders and DrQ, a data augmentation technique, were used to improve the performance.

An *autoencoder* is a combination of two neural networks, an encoder $f_{enc} : D \to K$ and a decoder $f_{dec} : K \to D$. Typically, the latent space D is smaller than the input dimension K. With the combination of the encoder and decoder, $z = f_{dec}(f_{enc}(x))$ can be computed. The goal of an autoencoder is that $z \approx x$. A reconstruction loss is used to train the autoencoder self supervised using gradient descent. For an image the reconstruction loss is given by the pixel wise mean square error. Figure 2.1 visualizes how an autoencoder works.

In RL, autoencoders can be used as an auxiliary loss to enhance the performance of agents trained on pixel observations. Yarats et al. [26] proposed an architecture where a shared CNN feature extractor is trained by the loss of the critic and an reconstruction loss. The feature extractor is also used by the actor network, but detached during training, i.e. no updates are performed on the encoder when training the actor.

Kostrikov, Yarats, and Fergus [12] investigated the effects of image augmentations on the performance of SAC pixels. Based on this, they developed a regularization approach called Data-regularized Q (DrQ) to enhance performance. Their idea was not simply to train the agent with augmented images but to average the Q-function over $M \geq 1$ different augmentations of the same image. Additionally, they averaged the Q target (equation 2.25) over $K \geq 1$ different augmentations. A differently positioned random cropping gives

the augmentations. All images from the environment have 84×84 pixels. First, they are padded by four pixels on each site. As padding, they used the repeated boundary pixels. Then a random crop of size 84×84 pixels is selected from the padded image.

3 Related work

This work combines three active fields of research in reinforcement learning (RL), Transfer Learning, Safe RL and the usage of pixel observations. The following sections will provide an overview of different approaches in these fields.

3.1 Transfer Learning

Transfer learning is very well known in supervised learning. It usually refers to using a pre trained model, or a part of it, and fine tune it on a new problem, reducing the training time on the new problem. This approach has been used for RL as well, for example by de la Cruz et al. [5]. They were able to improve the performance in some Atari games, by transferring some layers from agents, trained on other Atari games. Gamrian and Goldberg [7] found that an agent trained on the first level of the Nintendo game Road Fighter was not able to complete subsequent levels. However, when mapping the images back to the familiar images from the first level, using GANs, the agent was able to complete the level. Thus, they were able to transfer knowledge from one task to another.

3.2 Safe Reinforcement Learning

Gu et al. [8] defined five problems, called "2H3W", in safe RL. The problems are safety policy, complexity, applications, benchmarks, and challenges. Our work addresses the safety policy problem, formulated as "How can we perform policy optimization to search for a safe policy?"[8] Therefore, this section focuses on solutions for this problem.

Often, the problem is formulated as a Constrained Markov Decision Process (CMDP), as described in Section 2.1.1.1. From this, a Lagrangian function can be formulated, combining the reward function with the constraints. A number of Algorithms have been proposed to solve a CMDP with this approach [4, 27, 21, 15, 16]. Bharadhwaj et al. [3] combined this approach with a safety critic to estimate the expected cost, similar to our approach. However, they only use the critic during the sampling of actions. Others adapted trust region methods to also consider safety constraints [1].

A number of algorithms have been developed that attempt to train for reward and safety in different steps. Yang et al. [25] build upon TRPO to first learn a policy that optimizes the reward and then projecting it back to the closest policy that fulfills the constraints. Similarly, Wagener, Boots, and Cheng [23] developed an algorithm that is agnostic of the underlying RL algorithm. They learn a policy agnostic of safety, but before executing the actions in the environment, they are run through an intervention policy and potentially changed to ensure safety. A similar approach was followed by Pham, De Magistris, and

Tachibana [17]. They developed an OptLayer, that computes a safe action, closest to the original prediction. Wachi and Sui [22] proposed an algorithm that first expands a safety region, and then explores and exploits the region to optimize the reward.

3.3 Pixel Observations

Training RL agents on high dimensional image observations has proven to be less sample efficient and the agents are usually performing worse than their counterparts trained on the complete state information [26, 12, 18]. It is an active field of research, how to close the gap to state based agents.

Kostrikov, Yarats, and Fergus [12] investigated the effects of image augmentations on the performance and sample efficiency. Additionally, they proposed novel regularization techniques for the target Q and Q function, by averaging over multiple samples, obtained through different augmentations on a single observation.

Yarats et al. [26] used an autoencoder to aid the agents ability to learn a latent space from the pixel observations. Similarly, Srinivas, Laskin, and Abbeel [18] applied a contrastive loss as auxiliary loss to improve the encoder. To do so, they use two encoders, each getting different data-augmented versions of the observation. Based on this, Zhan et al. [28] proposed the FERM architecture, where a replay buffer is initialized with human demonstrations, which are then used to pre-train the encoder, using a contrastive loss. Afterwards, the pre-filled replay buffer and pre-trained encoder are used to train a SAC agent. Again, using contrastive loss, Stooke et al. [19] aided the training of the encoder by comparing it to an encoded example from k time steps later.

Lastly, when attempting to train under realistic assumptions, images are not the only possible observation. When using a robot arm, the position of the end-effector can be measured, as well as if its is open. A combination of images and a subspace of the complete state, regarding the robot, has been used by Kalashnikov et al. [11] to train robotic manipulation tasks.

4 Algorithm

The original goal of the preference reward algorithm was to transfer learned behavior from one RL agent to another. It is, however, designed to only rely on an abstract interface of the source of knowledge, the teacher. A teacher, as described in definition 4.1, if given a state, proposes an action together with a confidence about the quality of this action.

Definition 4.1 *Let $(S, A, P, R, \rho_0, \gamma)$ be a MDP. Then a function*

$$T : S \to A \times [0, 1]$$

is called a teacher *for this MDP. A teacher proposes an action for each state and gives a confidence about the quality of this action.*

In practice, a teacher does not necessarily has to be a RL agent, but can also be a trajectory planner or other means of solving the task at hand. See section 4.1.3 about different kinds of teachers. The following section 4.1 outlines the preference reward algorithm.

4.1 Preference Reward

The preference reward algorithm uses reward shaping to assist the exploration of a RL agent in training. It can therefore be used with any RL algorithm. The reward is changed based on the actions that a teacher suggests for the current state. This action is considered the preferred action. See section 4.1.3 for details about teachers.

Three different rewards are used for an MDP and a teacher T, the *environment reward*, *teacher reward* and *preference reward*. See definition 4.2. First, the reward from the MDP is called *environment reward* r_{env}. For a state s and an action a proposed by the agent, the teacher gives an action $\hat{a}$ and a confidence c. The confidence is used by the teacher to rate the quality of its action $\hat{a}$. The mean square error, of a and $\hat{a}$ is called the *action error*. The action error, multiplied with the negated confidence forms the *teacher reward* $r_{teacher}$. The teacher reward may be seen as a penalty for the agent if the proposed actions are not in line with the teachers actions. The confidence makes sure that the penalty is lower if the teacher is not confident about the quality of its action. Lastly, the weighted sum of the environment and teacher reward is called *preference reward* r_{pr}. The importance of the teacher reward is determined by a parameter $\alpha \in [0, 1]$.

Definition 4.2 *Let $(S, A, P, R, \rho_0, \gamma)$ be a MDP, T a teacher, (s, a, s') a state, action, next state triplet, and $\alpha \in [0, 1]$. Then:*

(1) $r_{env} = R(s, a, s')$ *is called* environment reward.

(2) Let $T(s) = (\hat{a}, c)$. *Then* $r_{teacher} = -c * \frac{1}{dim(A)} \sum_i (a_i - \hat{a}_i)^2$ *is called* teacher reward.

Algorithm 2: Preference Reward

Input: action $a \in \mathbb{R}^n$, state s, reward r_{env}, teacher T, $\alpha \in [0, 1]$
Data: $\hat{a} \in \mathbb{R}^n$, $c \in \mathbb{R}$, $r_{teacher} \in \mathbb{R}$
Output: preference reward $r_{pr} \in \mathbb{R}$

$\hat{a}, c \leftarrow T(s, a)$
$r_{teacher} \leftarrow -c * \frac{1}{n} \sum_{i=0}^{n} (a_i - \hat{a}_i)^2$
$r_{pr} \leftarrow \alpha * r_{teacher} + (1 - \alpha) * r_{env}$
return r_{pr}

(3) $r_{pr} = \alpha * r_{teacher} + (1 - \alpha) * r_{env}$ *is called* preference reward.

Algorithm 2 outlines the steps to calculate the preference reward. First, the teacher is used to obtain the preferred action $\hat{a}$ and the confidence c. Together with the action a from the agent, the teacher reward is calculated. Lastly, the preference reward is calculated from the environment reward and the teacher reward.

4.1.1 Adaptive Alpha

The parameter α that determines the importance of the preference reward does not necessarily need to be a fixed value. It is possible to change the value of α based on different factors. In fact, it turned out to increase the success rate of the agent, to reduce the α value as the training progressed. See the experiment results in section 5.3. We explored two methods for adapting α during training: one based on the number of training steps completed, and another based on the rewards our agent was receiving in its environment

For an adaptive α based on training steps, the α with a value of one and is linearly reduced to zero at the final training step. The adaptive α based on the environment reward assumes that the agent is getting better if the environment reward increases and therefore does not need as much guidance from the teacher. To make the α value more stable, we used an exponential moving average of the environment reward to calculate it.

Other options to adapt α are possible, but we leave them for future work.

4.1.2 High Dimensional Observations

In this work, the preference reward algorithm was used to assist the training of a SAC agent using high dimensional image observations, assuming that during training, the full state of the environment is available. Figure 4.1 shows the training setup. The SAC agent only relies on a frame stack as observations, the state is only needed for the teacher during training. Optionally, the gripper position can be added to the observation, to simplify the task. The gripper position could be obtained from a real world robot and does therefore not break the goal to train under realistic assumptions.

The idea behind this setup is to utilize more information during training than during evaluation. In a real world application, this would result in using more sensors during training to measure the state of the environment and use it for the teacher. But during

Algorithm 3: Ensemble Teacher

Input: action $a \in \mathbb{R}^n$, state s, list of RL agents L
Data: list of actions A
Output: action $\hat{a} \in \mathbb{R}^n$, confidence $c \in [0, 1]$

```
// L is only provided when initializing the teacher and not each time
```
foreach *agent in L* **do**
    A.APPEND (agent.GETACTION (s))
$\hat{a} \leftarrow$ SELECTRANDOMELEMENT(A)
$c \leftarrow$ EXP($-$STANDARDDEVIATION(A))
return $\hat{a}, c$

evaluation, the trained agent only relies on simple sensor data, such as images and the gripper position.

4.1.3 Teachers

As stated in definition 4.1, a teacher is a function that proposes an action for each state and provides a confidence about the quality of this action. For the training setup as described in section 4.1.2, the teacher may use a RL agent trained on the full state of the environment to obtain the preferred action. To obtain the confidence, algorithm 3 uses an ensemble of multiple teachers. First, the element wise standard deviation is calculated. Then the exponential of the negative standard deviation is used as confidence, to ensure a range of $[0, 1]$. From the actions proposed by the agents of the ensemble, one is drawn at random as the preferred action. Note that in this case, the teacher is not a mathematical function, as it is not deterministic. Another way to obtain the confidence with just a single agent is to use dropout as Bayesian approximation, as described in [6].

The training setup described in section 4.1.2 and figure 4.1 is not the only setup where a RL agent can be used as a teacher. In a slightly modified version of this setup, where the teacher and the student both receive the same observation, it is still possible to utilize an RL agent as a teacher. While the teacher would not be able to solve the task at hand perfectly, it could be an agent that can solve a simpler sub task and therefore guide the exploration of the agent in training.

However, the teacher does not necessarily has to be an RL agent. Other means to propose an action may be used as teachers as well. They only need to fulfill definition 4.1, that is being able to provide an action and a confidence for each state. One example would be a trajectory planner.

4.2 Safety

As described in section 2.1.1.1, a constrained Markov decision process is a MDP extended by a cost function $C(s, a, s') \rightarrow [0, 1]$ and a threshold $\delta \in \mathbb{R}_{\geq 0}$. Three different approaches were attempted in this work to extend SAC to CMDPs. The goal with these was not to

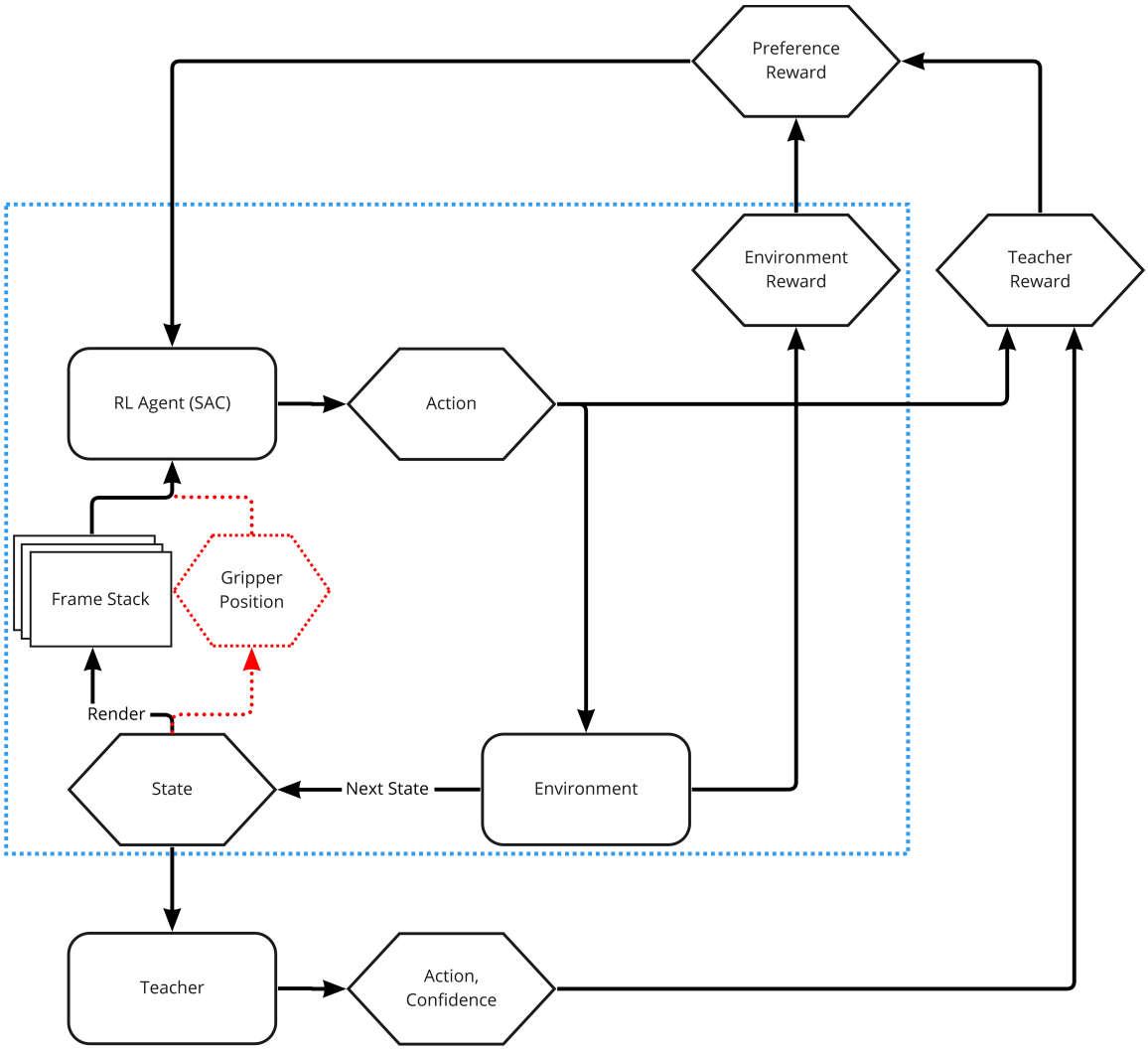


Figure 4.1: Overview of the preference reward algorithm to train an agent on high dimensional image observations. Optionally, the gripper position of a robot can be added to the observation. The elements marked in blue are present in regular SAC.

strictly stay below the threshold δ, but to learn a policy that minimizes the expected cost of an episode. One of the approaches uses reward shaping and the other two a safety critic.

4.2.1 Reward Based

The first, and simplest approach is to penalize the agent by deducting the cost of a state, action, next state triplet (s, a, s') from the reward:

$$R_{cost}(s, a, s') = \lambda_r R(s, a, s') - \lambda_c C(s, a, s') \tag{4.1}$$

The parameters $\lambda_r, \lambda_c \geq 0$ symbolize the importance of the reward and the cost. With this approach, the new reward function R_{cost} is not guaranteed to be in the interval $[0, 1]$.

4.2.2 Safety Critic

The cost is defined in the same way as the reward, the only difference is that the goal is to minimize the cost while the reward should be maximized. It is therefore possible to formulate the value functions and bellman equations $V^\pi(s)$ and $Q^\pi(s, a)$ just like in section 2.1.3. In the following, an index R or C is added to the functions to indicate whether they are the reward or safety function. The parameter vectors for the Q-function neural networks are indexed in the same way as ϕ_i^C and ϕ_i^R, where i is the index for the double-Q trick.

The functions for the optimal policy $V_C^*(s)$ and $Q_C^*(s, a)$ can not be formulated just like that, because the goal is not to minimize the cost but to maximize the reward while maintaining a low cost. It is therefore not clearly defined what the optimal policy is. The same is true for the reward function regarding the optimal policy $V_R^*(s)$ and $Q_R^*(s, a)$. This is, however, not a problem, since the optimal policy functions are not needed for SAC.

Together with the entropy, this results in the following cost Bellman equation similar to equation 2.23:

$$Q_C^\pi(s, a) = \mathop{\mathbb{E}}_{\substack{s' \sim P(\cdot|s,a) \\ a' \sim \pi}} \Big[C(s, a, s') + \gamma \Big(Q_C^\pi(s', a') - \alpha \log \pi(a'|s') \Big) \Big] \tag{4.2}$$

During training, the replay buffer also stores the cost of a transition. A sample from it is therefore a sextuple (s, a, r, c, s', d), where c denotes $C(s, a, s')$. With this, similar to equation 2.24, the safety-Q can be approximated:

$$Q_C^\pi(s, a) \approx c + \gamma (Q_C^\pi(s', a') - \alpha \log\ \pi(a'|s')),\ a' \sim \pi(\cdot|s') \tag{4.3}$$

Again, this is used to create the target function:

$$y_C(c, s', d) = c + \gamma(1 - d) \Big(\max_{j=1,2} Q_{\overline{\phi}_i^C}(s', a') - \alpha \log\ \pi_\theta(a'|s') \Big),\ a' \sim \pi_\theta(\cdot|s') \tag{4.4}$$

Just like for the reward critic, older versions of the two Q-functions are used to calculate the target. This stabilizes the training. They are denoted with $\overline{\phi}_i^C$. Note that in contrast to

equation 2.25 the maximum of the two Q-functions is used. Just like the reward critic, the safety critic can be trained with:

$$J_{Q_C}(\phi)_i = \mathop{\mathbb{E}}_{(s,a,r,c,s',d)\sim D}\left[\left(Q_{\phi_i^C}(s,a) - y_C(c,s',d)\right)^2\right] \tag{4.5}$$

In this work, two different ways to use the safety critic were evaluated. One is to use the safety critic during the training of the policy. The other approach only uses the safety critic in the evaluation.

4.2.2.1 Safety Training

The first approach using the safety critic changes the training objective of the actor. Instead of training an actor that is proportional to the exponential of the Q-function, it should now be proportional to the difference between the reward and safety critic:

$$\pi(a|s) \propto \exp(Q_R(s,a) - Q_C(s,a)) \tag{4.6}$$

This also changes the goal to minimize the Kullback-Leibler divergence between the actor and the exponential difference between the two critics:

$$\mathop{\mathbb{E}}_{s\sim D}\left[D_{KL}\left(\pi_\theta(\cdot|s)\middle\|\frac{\exp\left(\frac{1}{\alpha}(Q_{\phi^R}(s,\cdot) - Q_{\phi^C}(s,\cdot))\right)}{Z_\phi(s)}\right)\right] \tag{4.7}$$

$$= \mathop{\mathbb{E}}_{s\sim D}\left[\sum_{a\in A}\pi_\theta(a|s)\log\left(\frac{\pi_\theta(a|s)Z_\phi(s)}{\exp\left(\frac{1}{\alpha}(Q_\phi^R(s,a) - Q_{\phi^C}(s,\cdot))\right)}\right)\right] \tag{4.8}$$

$$= \mathop{\mathbb{E}}_{\substack{s\sim D\\ a\sim\pi_\theta(\cdot|s)}}\left[\log\pi_\theta(a|s) + \log Z_\phi(s) - \frac{1}{\alpha}(Q_{\phi^R}(s,a) - Q_{\phi^C}(s,\cdot))\right] \tag{4.9}$$

$$= \mathop{\mathbb{E}}_{\substack{s\sim D\\ a\sim\pi_\theta(\cdot|s)}}\left[\alpha\log\pi_\theta(a|s) + \alpha\log Z_\phi(s) - Q_{\phi^R}(s,a) + Q_{\phi^C}(s,\cdot)\right] \tag{4.10}$$

Just like it is done for regular SAC in equation 2.30, the term $Z_\phi(s)$ can be omitted and the equation is multiplied by α to get the new training objective:

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{\substack{s\sim D\\ a\sim\pi_\theta(\cdot|s)}}\left[\alpha\log\pi_\theta(a|s) - Q_{\phi^R}(s,a) + Q_{\phi^C}(s,\cdot)\right] \tag{4.11}$$

Again similar to regular SAC, the reparametrization trick from equation 2.32 has to be used. Additionally the double Q-trick is added for both critics, resulting in the following objective:

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{\substack{s\sim D\\ \xi\sim\mathcal{N}(0,I)}}\left[\alpha\log\pi_\theta(f_\theta(s,\xi)|s) - \min_{i=1,2}Q_{\phi_i^R}(s,f_\theta(s,\xi)) + \max_{i=1,2}Q_{\phi_i^C}(s,f_\theta(s,\xi))\right] \tag{4.12}$$

Note that in order to use the more pessimistic of the two safety critics, the maximum of the two has to be used. This leads to some changes in the SAC algorithm, as described in algorithm 1. In addition to the updates to the reward critic, now at ever step also the safety critic is updated. To do so, also the safety critic targets need to be updated. Finally, the actor is trained with the new objective from equation 4.12. The new algorithm is described in detail in algorithm 4

Algorithm 4: Safety SAC

Input: entropy coefficient α, environment e, initial parameters $\theta, \phi_1^R, \phi_2^R, \phi_1^C, \phi_2^C$, learning rates η_π, η_Q, target update rate ρ

Data: replay buffer D

Output: $\theta, \phi_1^R, \phi_2^R, \phi_1^C, \phi_2^C$

`// Initialize target networks`

$\overline{\phi}_1^R \leftarrow \phi_1^R, \overline{\phi}_2^R \leftarrow \phi_2^R$

$\overline{\phi}_1^C \leftarrow \phi_1^C, \overline{\phi}_2^C \leftarrow \phi_2^C$

$s \leftarrow e.\textsc{initialState}()$

for *each iteration* **do**

`// Interact with the environment`

$a \sim \pi_\theta(\cdot|s)$

$r, c, s', d \leftarrow e.\textsc{step}(a)$

$D.\textsc{append}((s, a, r, c, s', d))$

if $d = done$ **then**

$e.\textsc{reset}()$

$s \leftarrow e.\textsc{initialState}()$

Sample a batch B of transitions from D

`// Compute Q-function targets`

foreach $(s, a, r, c, s', d) \in B$ **do**

$$y_R(r, s', d) \leftarrow r + \gamma(1-d)\left(\min_{j=1,2} Q_{\overline{\phi}_i^R}(s', a') - \alpha \log\ \pi_\theta(a'|s')\right),\ a' \sim \pi_\theta(\cdot|s')$$

$$y_C(c, s', d) \leftarrow c + \gamma(1-d)\left(\max_{j=1,2} Q_{\overline{\phi}_i^C}(s', a') - \alpha \log\ \pi_\theta(a'|s')\right),\ a' \sim \pi_\theta(\cdot|s')$$

`// Update Q-functions and policy by one step of gradient descent`

for $i = 1, 2$ **do**

$$\phi_i^R \leftarrow \phi_i^R - \eta_Q \nabla_{\phi_i^R} \frac{1}{|B|} \sum_{(s,a,r,c,s',d)\in B} \left[\left(Q_{\phi_i^R}(s, a) - y_R(r, s', d)^2\right)\right]$$

$$\phi_i^C \leftarrow \phi_i^C - \eta_Q \nabla_{\phi_i^C} \frac{1}{|B|} \sum_{(s,a,r,c,s',d)\in B} \left[\left(Q_{\phi_i^C}(s, a) - y_C(r, s', d)^2\right)\right]$$

for $i = 1, 2, \ldots, |B|$ **do**

$\xi_i \leftarrow \mathcal{N}(0, I)$

$$\theta \leftarrow \theta - \eta_\pi \nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left[\alpha \log \pi_\theta(f_\theta(s, \xi)|s) - \min_{i=1,2} Q_{\phi_i^R}(s, f_\theta(s, \xi)) + \max_{i=1,2} Q_{\phi_i^C}(s, f_\theta(s, \xi))\right]$$

`// Update target networks`

for $i = 1, 2$ **do**

$$\overline{\phi}_i^R \leftarrow \rho\overline{\phi}_i^R + (1-\rho)\phi_i^R$$

$$\overline{\phi}_i^C \leftarrow \rho\overline{\phi}_i^C + (1-\rho)\phi_i^C$$

return $\theta, \phi_1^R, \phi_2^R, \phi_1^C, \phi_2^C$

Algorithm 5: Safety Evaluation

Input: state s, actor A, reward critic Q_R, safety critic Q_C, number of samples n, safety threshold δ
Data: list of action, cost tuples L
Output: action a

do n **times**
  $a \leftarrow A.\text{SAMPLEACTION}(s)$
  $c \leftarrow Q_C(s, a)$
  $L.\text{APPEND}((a, c))$
if $\exists (a, c) \in L : c < \delta$ **then**
  $L \leftarrow \{(a, c) \in L | c < \delta\}$
  $(a, c) \leftarrow \underset{(a,c)\in L}{\arg\max}\, Q_R(s, a)$
  return a
else
  $(a, c) \leftarrow \underset{(a,c)\in L}{\arg\min}\, c$
  return a

4.2.2.2 Safety Evaluation

The second approach uses the safety critic only during the evaluation. The safety critic is trained alongside the reward critic, but only the reward critic is used to train the actor. The agent does not consider the cost of the actions during training. It is, therefore, expected to produce high cost actions during training. This makes this approach only viable for tasks where, during training, neither the actor nor the environment is likely to break due to high cost actions.

When the agent is evaluated, or used in practice, the trained safety critic comes into play. SAC trains a stochastic actor. However, a deterministic version of the actor is usually used for the evaluation. This can be achieved by not applying any randomization in the reparametrization trick and directly returning $\mu_\theta(s)$ as the action. See equation 2.32 for reference. The safety evaluation algorithm relies on the stochastic nature of a SAC actor. Therefore, the stochastic version of the actor is used during evaluation when using the safety evaluation algorithm.

The safety evaluation algorithm is described in algorithm 5. Its goal is to select an action for a state that has a low cost. The stochastic actor makes it possible to sample n different actions for a state. For each action, the expected cost is calculated with the safety critic. First, it is checked if any of the actions has an expected cost of less than a threshold δ. If this is not the case, the action with the lowest expected cost is returned. If some actions suffice the cost threshold δ, the expected reward for each of them is calculated using the reward critic. The action with the highest expected reward is returned.

4.3 Implementation Details

Some aspects of the preference reward algorithm were slightly altered in the implementation when compared to section 4.1. In definition 4.2(2) the teacher reward $r_{teacher}$ is defined as the mean square error between the action of the student and the teacher multiplied by the confidence. The environments used in the evaluation of this work all have an action space of $[-1, 1]^4$. This leads to $r_{teacher}$ being in range $[0, 4]$. To scale $r_{teacher}$ to the same range as the environment reward r_{env}, it is divided by the maximum mean square error possible for the action space, in this case four.

As stated above, the action space has four dimensions. The first three dimension are for the movement in the three dimensional space. The last dimension is used to control the opening of the gripper. See section 5.1 for a description of the environments used. The three environments used in this work all ignore the value given to control the opening of the gripper. The fourth dimension is only present to make the environments compatible with others that have the robot pick up the object. To account for the unused fourth dimension, the mean square error in the teacher reward is only calculated over the first three dimensions.

When training on high dimensional image observations, the observations are fed through a CNN encoder, to map them into a smaller latent space. This latent space is then given to the actor and critic networks. The CNN is only trained using the reward critic. The actor and the safety critic use the same CNN as the reward critic for the forward pass, but it is decoupled for the backpropagation.

The implementation can be found under `https://github.com/f-krone/SafeTransferLearningInChangingEnvironments`.

5 Experiments

This chapter describes the experiments conducted for this work. First, the environments used for the experiments are described. Then the setup and goal is explained. And lastly, the results are shown and evaluated.

5.1 Environments

Two of the environments used in this work come from the OpenAI gym robotics tasks. These are the FetchReach and the FetchPush environments. The third environment, called FetchPushBarrier, is a modification of the FetchPush environment. See figure 5.1 for an overview. All three environment share the same setup. The robot arm can be controlled with actions $a \in [-1, 1]^4$. The first three dimensions of the vector control the movement of the arm along the axes. The fourth dimension allows to open and close the gripper. It is, however disabled for these environments to open the gripper. A value has to be provided nonetheless, but its value is ignored.

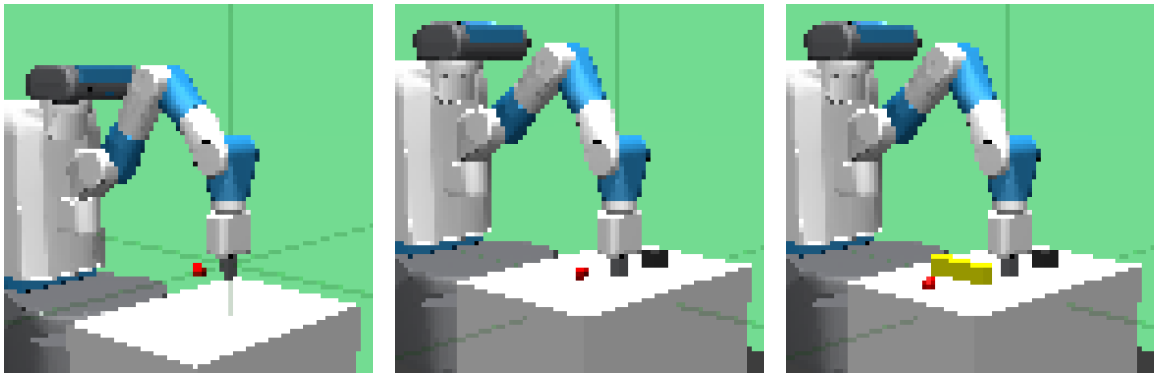
The goal in the FetchReach environment is to move the gripper to the red sphere. The FetchPush and FetchPushBarrier environments share the same goal of moving the black cube to the red sphere. The FetchPushBarrier environment has the added difficulty that the yellow barrier has to be avoided. The red sphere in all environments is only a marker for the target and has no physical body.

The FetchPush environment is slightly modified with an increased number of steps. The maximum is now 100 instead of the regular 50. This gives the agent room for small mistakes when moving the object.

5.1.1 FetchPushBarrier

The FetchPushBarrier environment is an adaptation of the regular FetchPush environment. The only things changed are the initialization to add a barrier and a cost function.

When initializing the FetchPushBarrier environment, first the gripper and the object are seeded the same as in the FetchPush environment. To position the barrier, first a random position along the x-axis is chosen. See figure 5.2 for the naming of the axes. The position is drawn to let the barrier extend to the table edges at the most. To sample the placement along the y-axis, first it has to be determined if the barrier should be in front or behind the object, if seen from the camera. If the object is at the very front of the table, the barrier is always placed behind the object. If the object is at the very back, it is placed in front respectively. If the object is in the middle of the table, it is chosen at random if the barrier should be placed in front or behind the object. Afterwards, the exact placement along the



(a) FetchReach (b) FetchPush (c) FetchPushBarrier

Figure 5.1: Overview of the three environments used in this work. The images show the actual 84×84 pixel observations the agents receive. The observations are a cropping of the original image, the gym Fetch environments render. See figure 5.2 for the original image size.

y-axis is drawn at random, ensuring a minimum distance to the object, the gripper and the table edge. The size of the barrier is fixed and it is always position parallel to the axes.

After the barrier is placed, the position of the target is drawn at random. The placement is constrained in the y-axis to always be behind the barrier if seen from the object. In the x-axis, the position is constrained to the size of the barrier, to ensure that the object has to be moved around the barrier.

When taking a step in the environment, the reward is calculated just like in the FetchPush environment. For a dense reward, the negative distance between the object and the target is used. The sparse reward is 0 if the distance between the goal and the target is below a threshold and -1 otherwise. Additionally a cost is computed after each step:

$$C(s, a, s') = \begin{cases} 0, & \text{if } d \geq 0.1 \\ 1 - \frac{d}{0.1}, & \text{else} \end{cases} \tag{5.1}$$

Where d is given by the minimum of the distances of the gripper and object to the barrier.

The maximum steps in the environment is increased to 150, to account for the added difficulty.

5.2 Experiment Setup

The goal with the experiments was to train an agent using frame stack observations for all three environments described in section 5.1. The experiments use the training setup described in section 4.1.2 with an ensemble teacher. This also includes the described option of augmenting the frame stacks with the gripper position to simplify the training. The high dimensional image observations consist of a stack of three frames. Each frame is an image of 84×84 colored pixels.

First, for each environment, an ensemble of three SAC agents using the state was trained. If one of the agents was significantly worse than the others, more agents were trained and

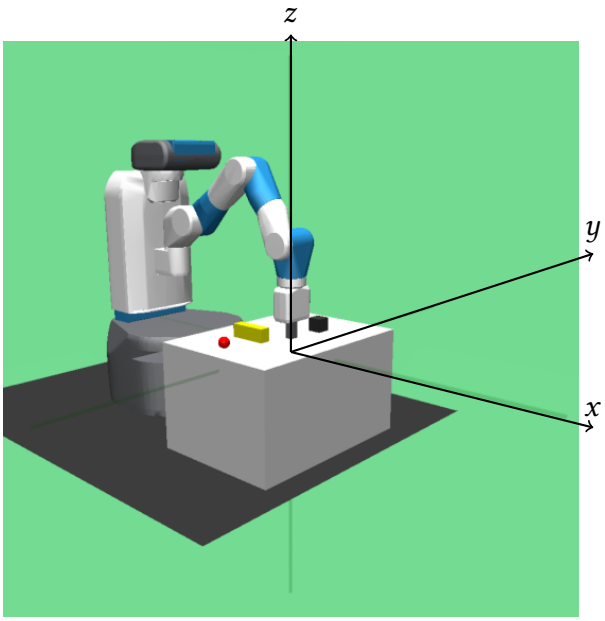


Figure 5.2: Axes in the FetchPushBarrier environment

the best three picked for the ensemble. See table 5.1 for the exact results of the ensemble agents. Afterwards, the agents using frame stacks or augmented observations are trained with the ensemble. All environments use a dense reward, i.e. a reward is not only given in the case of a success, but at each time step.

The machines used for training either used an Intel Core i9-9900K with 64 GiB of memory and a Nvidia RTX 2080 TI or an AMD Ryzen Threadripper 3960X with 130 GiB of memory and a Nvidia RTX A5000. CUDA version 11.6 was used with both GPUs.

5.3 Results

The following section contains the results of the experiments made during this work. Experiments were made in the environments FetchReach, FetchPush and FetchPushBarrer. The most experiments were made in the FetchReach environment. Training setups that do not perform well in this environment are expected to not have meaningful results in the much harder FetchPush and FetchPushBarrier environments. Therefore some setups were omitted for these environments. The results shown in this section are, if not otherwise stated, averaged over 3 trainings with different random seeds.

5.3.1 FetchReach

The FetchReach environment is fairly easy and was therefore chosen as a starting point for the experiments in this work. As table 5.1 shows, the three agents of the ensemble for the teacher are all able to solve the task each time. The agents using image observations were trained with regular SAC and SAC together with the two improvements described in section 2.2, autoencoders and DrQ. All algorithms were trained with only frame stacks

Environment	Success Rate	Episode Reward	Episode Cost	Mean Success Rate	Mean Reward	Mean Cost
FetchReach	1.0	-0.600		1.0	-0.552	
	1.0	-0.572				
	1.0	-0.484				
FetchPush	0.930	-3.374		0.901	-4.134	
	0.849	-4.774				
	0.925	-4.253				
FetchPushBarrier	0.726	-14.873	1.370	0.761	-14.083	1.58
	0.790	-12.439	1.075			
	0.767	-14.937	2.294			

Table 5.1: Overview of the results from all ensembles used in training. All agents were evaluated over 1000 episodes. The same seed was used for all environments.

and frame stacks augmented with the gripper position. Additionally, all algorithms were trained with and without the teacher. The α value that defines the importance of the teacher is set to one. This results in a total of 12 combinations. The success rates during training of all 12 setups are plotted in figure 5.3. The success rate of SAC state is added as reference. The episode rewards during training are plotted in figure 5.4. It does, however, not make sense to compare the rewards of the setups with and without a teacher, since the preference reward algorithm uses reward shaping. Additionally, all agents were evaluated on 1000 episodes. The mean success rate and reward can be seen in table 5.2.

The training setups using DrQ were all able to reliably solve the task and they are also the only ones to do so. Adding the gripper information to the observations did, however, increase the sample efficiency. The same is even more true for adding a teacher. Combined, this results in the DrQ agent using augmented states and a teacher almost being as efficient as SAC state. The mean rewards from the evaluation show a similar pattern. The reward is always higher when adding a robot or a teacher. Since the environment uses dense rewards, i.e. the distance between the gripper and the target at each time step, this indicates that the agents are able to solve the task faster.

For the setups using the SAC and SAC-AE algorithms, the picture is not quite as clear. Plain SAC does not benefit from adding the gripper information. Adding a teacher, however, significantly increases the success rate and sample efficiency. SAC-AE benefits from the gripper information and the teacher. Using the teacher results in a higher success rate than adding the gripper information. However, if the gripper information is added to the observations, also adding the teacher does not further increase the success rate. The success rate is actually lower that using SAC-AE with a teacher but without the gripper information. A possible explanation for this lies in the architecture of the SAC-AE agent. The autoencoder only uses the frame stack and tries to reproduce it. It does not benefit from the added gripper information. It also does not benefit from the reward shaping through the teacher, as the autoencoder uses a reconstruction loss and is independent from the reward. This is in contrast to the actor and critic that both benefit from the gripper

Algorithm	Robot	Teacher	Mean Success Rate	Mean Episode Reward
drq	False	False	0.994	-0.852
		True	0.994	-0.720
	True	False	0.993	-0.763
		True	0.996	-0.698
sac	False	False	0.425	-3.360
		True	0.608	-2.945
	True	False	0.393	-3.622
		True	0.628	-2.912
sacae	False	False	0.388	-3.484
		True	0.657	-2.787
	True	False	0.573	-2.882
		True	0.588	-3.084

Table 5.2: Mean success rate and reward for all agents trained on the FetchReach environment. The shown values are the mean over the three agents trained for each setup. All agents were evaluated on 1000 episodes. The environments were initialized with the same seed. The preference reward algorithm is not used during evaluation. Therefore, the reward shown here is the environment reward.

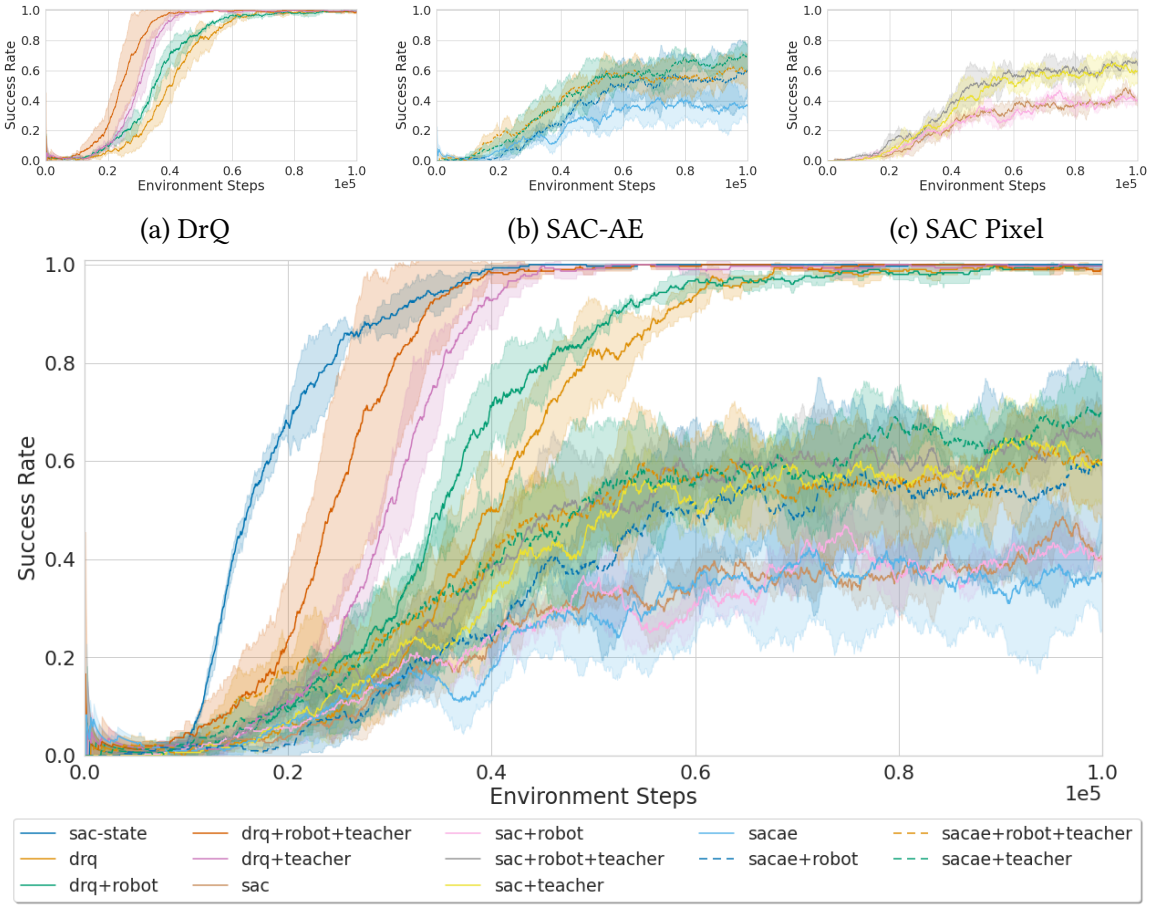
information and the teacher. This difference could lead to very different updates on the shared CNN encoder during training and could explain the discrepancy in the success rate. The fact that the SAC-AE+robot+teacher setup is outperformed by the same setup without the autoencoder supports this theory.

When looking at the episode reward in figure 5.4, it is clear, that DrQ has the highest reward in all setups. This is consistent with the success rate of the DrQ setups. Also, it can be seen that when using a teacher, also adding the gripper information to DrQ increases the sample efficiency. The episode rewards for the setups using plain SAC and SAC-AE are very similar. Again, this is consistent with the success rates. The success rates of the SAC and SAC-AE setups with a teacher are fairly similar to each other. The same is true for the setups without a teacher.

In general, it can be said that adding a teacher always lead to improvements in the success rate or the sample efficiency over the base version of the algorithm. In fact, for all three algorithms, the best training setup uses a teacher.

5.3.2 FetchPush

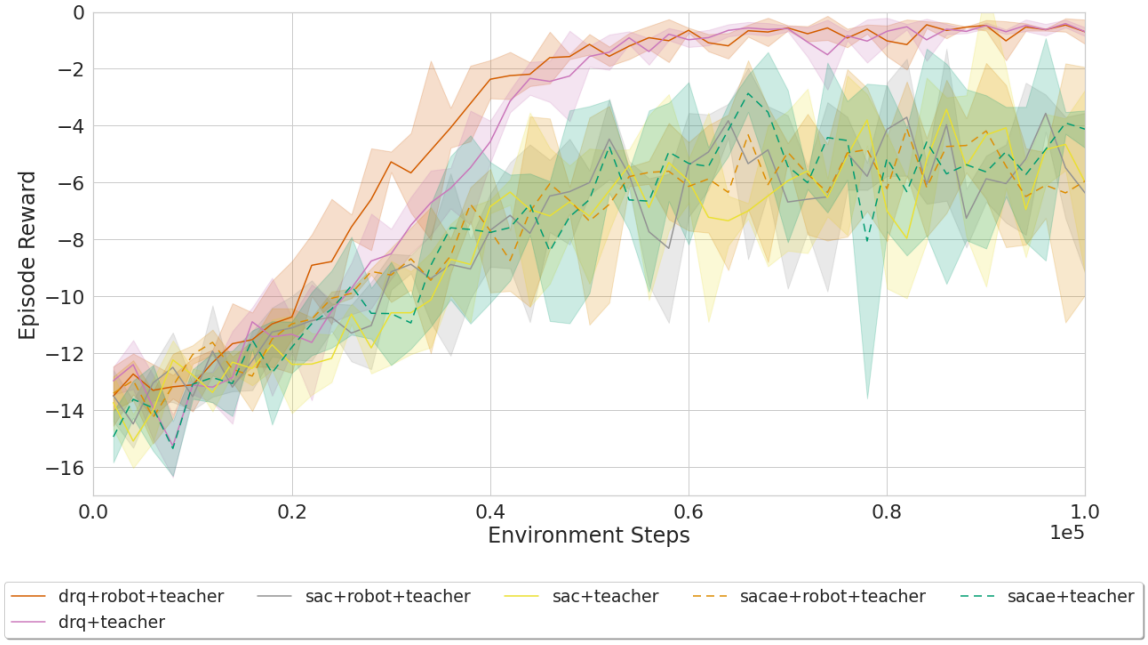
Only agents trained with DrQ were able to reliably solve the FetchReach environment. Since the FetchPush environment is significantly harder than the FetchReach environment, all agents were trained with DrQ. Additionally, the robot state augmentations were used for all experiments. Again, this is based on the experience with FetchReach, where this improved the sample efficiency.



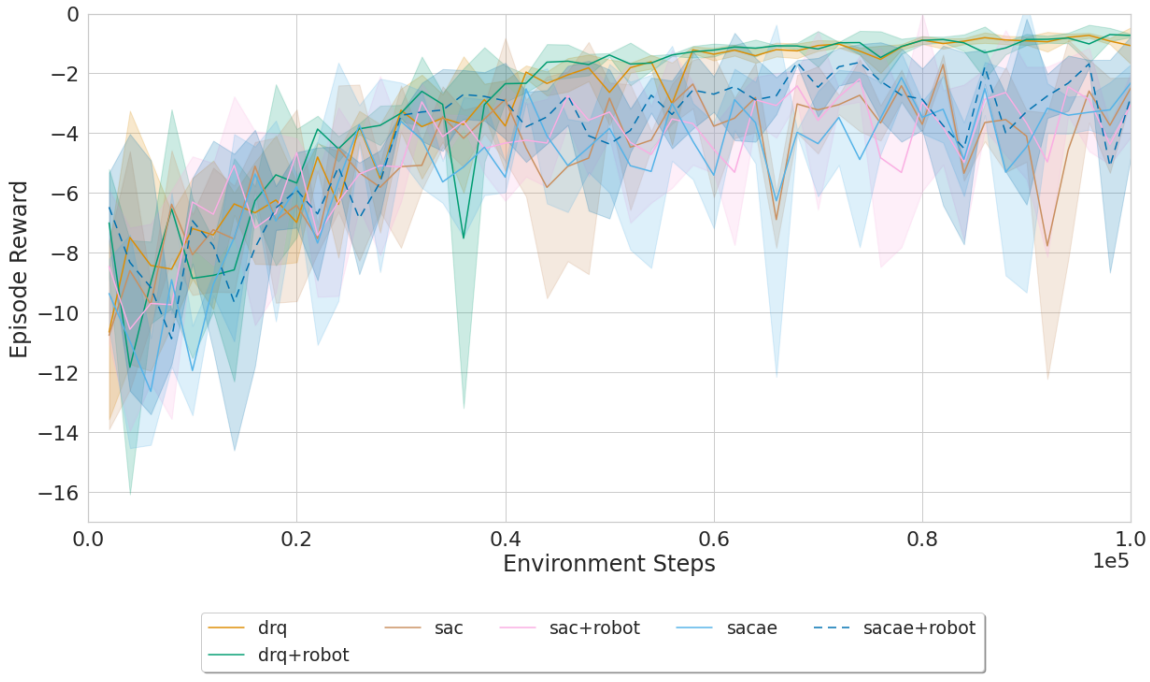
(a) DrQ (b) SAC-AE (c) SAC Pixel

(d) All FetchReach experiments

Figure 5.3: The success rates from all experiments on the FetchReach environment. The algorithms SAC pixel, SAC autoencoder and DrQ were each trained with and without augmented states and teachers. As a reference, SAC state is added. The subplots (a)-(c) each show the success rates for one algorithm.



(a) Episode reward with a teacher



(b) Episode reward without a teacher

Figure 5.4: The episode rewards from all experiments on the FetchReach environment. The algorithms SAC pixel, SAC autoencoder and DrQ were each trained with and without augmented states and teachers. Since the preference reward algorithm uses reward shaping, it does not make sense to compare the experiments with and without a teacher.

Experiments were conducted with and without a teacher. When a teacher was used, the fixed α values of 1.0 and 0.5 were tried. Additionally, an adaptive α was used as described in section 4.1.1. The adaptive α is based on the environment reward. To stabilize the adaptive α value, it is calculated from a moving average of the environment reward for one episode with a coefficient of 0.01. The α value was then given by the following function:

$$\alpha(r_{env}) = \begin{cases} 1, & \text{if } r_{env} < -20 \\ 1 - \frac{-20 - r_{env}}{-20}, & \text{else} \end{cases}$$

With this function, the α value starts at one at the beginning of the training and gets lower once the agent surpasses an environment reward of -20 per episode. At a non achievable environment reward of zero, the α value would also reach zero.

Figure 5.5 shows the success rates during training of all experiments. The different α values of the experiments with a teacher and their development during training are plotted in figure 5.6. The rewards during training are not shown here, as all experiments use different α values and the rewards are therefore not comparable. All trained agents were evaluated on 1000 episodes. The mean success rates and episode rewards can be seen in table 5.3.

When looking at the success rate of the experiment without a teacher, it is clearly visible that there is no noticeable training effect. The success rate does not change over the course of the training. The way the FetchPush environment is initialized, it is possible for the object and the target to touch each other from the beginning. The environment is therefore sometimes solved without the robot moving the object. Most likely this is why the success rate is not zero when training without the teacher.

When training with a fixed α of one, the agents were able to solve the task in roughly 40% of the cases during training. When evaluated over 1000 episodes, the mean success rate was 40.3%. The success rate of the agents trained with an α of 0.5 increased a little slower that the one with an α of one, but surpassed it at some point. During training, a success rate of up to 70% was reached. However, the success rate shows some fluctuation. In the final evaluation, the success rate was 61.5%. During training, the success rate is calculated based on the last 100 episodes. The final evaluation is calculated using 1000 episodes. This explains the difference in the results. A possible explanation for the higher success rate with a lower α lies in the quality of the teacher. As it can be seen in table 5.1, the ensemble, the teacher consists of, has a mean success rate of 90.1%. With an α of one, the agent in training can only learn to imitate the teacher. If, however, the teacher is not acting perfectly, the agent will learn subpar actions. With a lower α value, the agent gets direct feedback from the environment an can also learn from this.

With the adaptive α we saw a success rate very similar to the experiments with a fixed α of 0.5. Figure 5.6 shows the development of the α value over the course of the training. The adaptive value dropped significantly below the fixed value of 0.5. However, this did not lead to changes in the success rate. In the final evaluation, we see a slight improvement of the agents with an adaptive α over the ones with a fixed α of 0.5, with regards to both the success rate and the mean episode reward.

One problem that might lower the performance with the adaptive α lies in the architecture of Off-policy RL algorithms in combination with the preference reward algorithm.

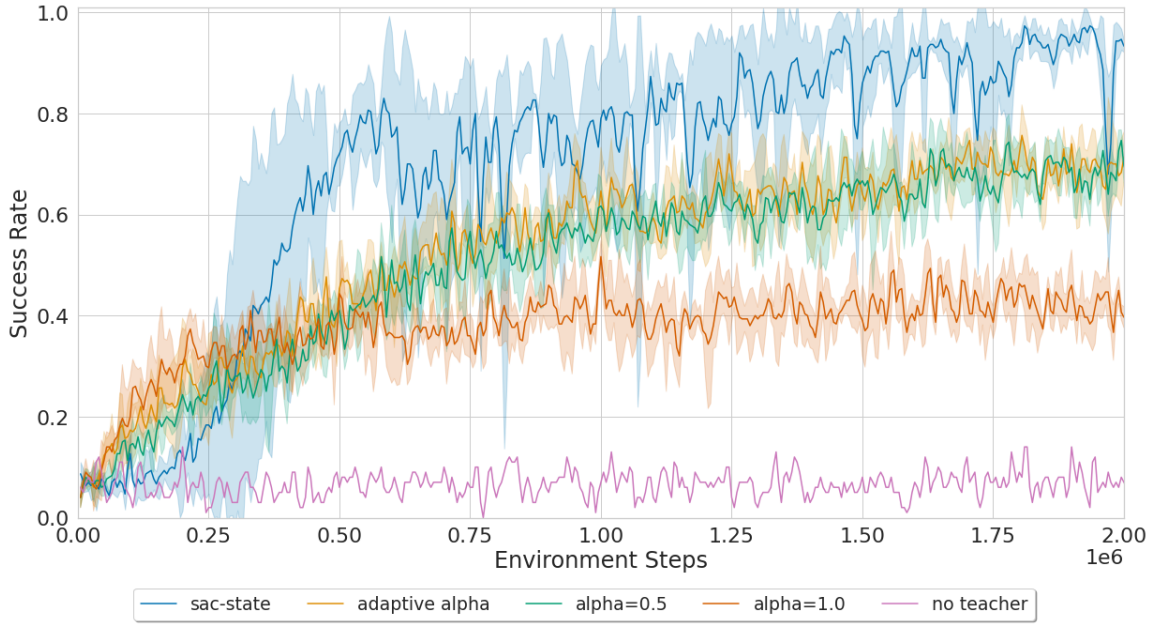


Figure 5.5: The success rates from all experiments on the FetchPush environment. All agents were trained with DrQ and the robot state augmentations. As a reference, SAC state is added. In contrast to the other experiments, the agent without a teacher was only trained with one seed. It is however clearly visible that there is no noticeable training effect.

While both the environment reward r_{env} and the teacher reward $r_{teacher}$ are in range $[0, 1]$, it is possible that $r_{teacher}$ is significantly higher than r_{env}. This can happen especially if r_{env} is very small for each step. When r_{env} increases and the α is reduced, this can lead to overall lower values of the preference reward r_{pr}, although the agent is performing better. Since the replay buffer only stores r_{pr}, the agent then learns the wrong behavior. A possible solution for this would be to store both r_{env} and $r_{teacher}$ and only calculate r_{pr} when it is needed for the training, using the current α value. However, this possible improvement was left for future works, due to time restricitions.

Overall, the adaptive α slightly improved the performance. However, the main benefit is that it gives us a different way to set the importance of the teacher. It can be hard to set a good α value. With the adaptive α, we instead have to set a reward threshold when to start lowering the α value. Note that this is only one possibility to adapt the α value. Other functions may be used as well.

5.3.3 FetchPushBarrier State

As described in section 5.1.1, the FetchPushBarrier environment is an adaptation from the FetchPush environment that adds a cost function. To represent the cost, the three methods from section 4.2, reward based, safety training, and safety evaluation, were used. The parameters λ_r and λ_c that define the importance of the reward and the cost in the

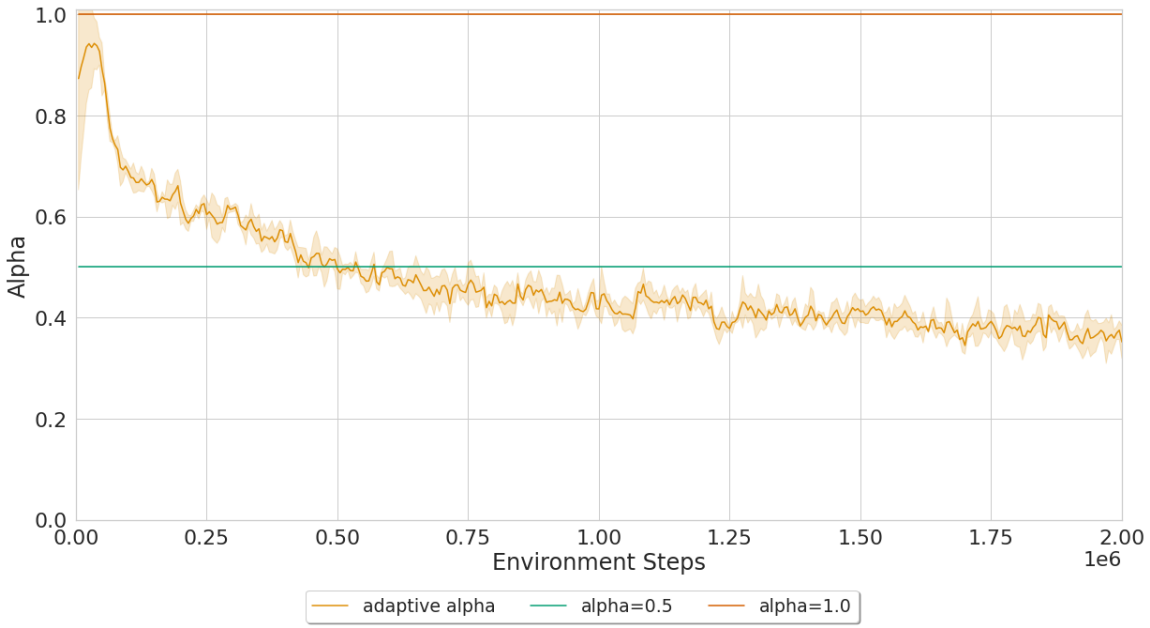


Figure 5.6: The α value from all experiments on the FetchPush environment. The α value of the experiments with an adaptive α is calculated with a function based on the environment reward. See section 4.1.1 for reference.

Experiment	Mean Success Rate	Mean Episode Reward
Adaptive α	0.617	-7.641
α=0.5	0.615	-8.338
α=1.0	0.403	-11.269
No teacher	0.071	-17.316

Table 5.3: Mean success rate and reward for all agents trained on the FetchPush environment. The shown values are the mean over the three agents trained for each setup. All agents were evaluated on 1000 episodes. The environments were initialized with the same seed. The preference reward algorithm is not used during evaluation. Therefore, the reward shown here is the environment reward.

reward based approach were both set to one. All agents were trained using the state space as observations.

Figure 5.7 shows the success rate and figure 5.8 the cost during training. To show the effect of the safety evaluation, the agents were evaluated every 25000 training steps. The figures 5.8a and 5.8b show the cost during training and periodic evaluation respectively. For the periodic evaluation, the safety evaluation was used with the parameter $n = 10$ and $\delta = 0.05$. The parameter n defines how many actions are sampled from the policy for each action that is needed from the agent. The parameter δ defines the threshold when to select the action that is expected to maximize the reward or minimizes the cost. Additionally, all agents were evaluated on 1000 episodes. The results are shown in table 5.4. The values are averaged over the three agents trained with each algorithm.

When looking at the success rate, all algorithms show a similar sample efficiency. However, the safety training shows a higher success rate than the reward based approach. The safety evaluation performed worse than the reward based approach. In the final evaluation, the safety training and the reward based approach have very similar success rates of 68% and 67.5%.

The cost during training, as plotted in figure 5.8a shows a similar picture. Again, the safety training performs the best, followed by the reward based approach. The high cost from the safety evaluation was expected, as only the reward is optimized during training with the safety evaluation. In the final evaluation, the safety training clearly outperforms the reward based approach with a mean cost of 1.757 against 3.115. With the safety training there is also a clearly visible training effect with regards to the cost. The cost gets lower, the further the training progressed. With the reward based approach, the training effect is not quite as visible. This might be due to the fact that the final cost is not as good. When looking at the cost during the periodic evaluation, the safety evaluation clearly lowered the costs, but is still significantly worse than the two other approaches. Also the cost seem to be very similar over the course of the training after a short period with lower costs. This could be explained with the way the FetchPushBarrier environment works. To produce costs, the agent first needs to learn to move the object towards the target, since the barrier is always located between the object and the target. As the agent improves, it will more often come into states were costs are possible, mitigating the effect of the improved avoidance of costs.

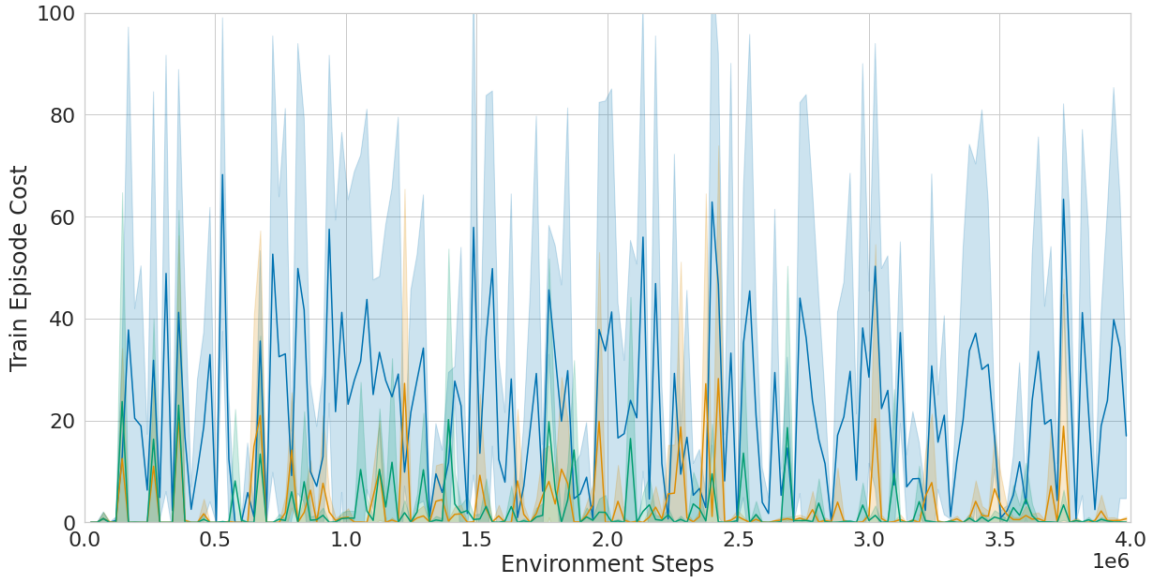
The three agents trained for the safety evaluation were additionally evaluated with different values for n and δ. For n, the values 1, 2, 5, and 10 were used. For δ the values 0.05, 0.1, 0.2, and 0.5. Not that for $n = 1$ the choice of δ does not matter. Each agent was evaluated for 1000 episodes with each parameter combination. Table 5.5 shows the mean success rate, episode reward and episode cost for each parameter combination. The mean is calculated over the results from the three agents. Additionally, the results are visualized in figure 5.9.

It is clearly visible that a higher value for n results in a lower cost. This is easily explained with the way the algorithm works. The algorithm samples n actions from the actor. With a higher value for n, the probability to get a low cost action is higher. However, a higher value also results in a longer execution time. Therefore, the parameter n has to be chosen with a trade-off between cost and execution time. Drawing more sample also slightly increased the success rate. This can be explained by the stochastic nature of the actors.

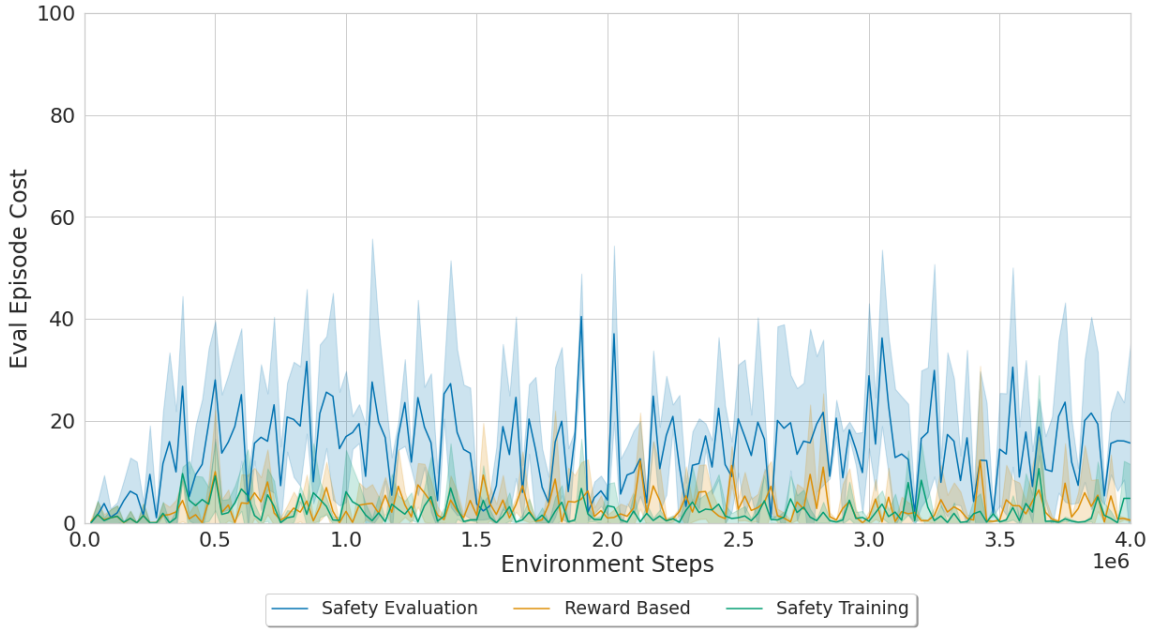
Figure 5.7: The success rates from the FetchPushBarrier state experiments. The FetchPush-Barrier environment was trained with the reward based approach as well as the safety training and safety evaluation algorithm.

Algorithm	Mean Success Rate	Mean Reward	Mean Cost
Reward Based	0.675	-16.466	3.115
Safety Evaluation deactivated	0.551	-15.710	21.057
Safety Evaluation activated	0.576	-15.498	16.069
Safety Training	0.680	-13.278	1.757

Table 5.4: Mean success rate and reward for all agents trained on the FetchPushBarrier environment using state. The shown values are the mean over the three agents trained for each setup. All agents were evaluated on 1000 episodes. The safety evaluation agents were evaluated with the parameters $n = 10$ and $\delta = 0.05$. This choice of parameters resulted in the lowest mean episode cost with the safety evaluation.



(a) Episode cost during training



(b) Episode cost from periodic evaluation during training

Figure 5.8: The episode cost from the FetchPushBarrier state experiments. The FetchPush-Barrier environment was trained with the reward based approach as well as the safety training and safety evaluation algorithm. During the training, the agents were periodically evaluated. Subplot (a) shows the costs during training and subplot (b) the cost during the evaluation. The safety evaluation used the parameters $n = 10$ and $\delta = 0.05$.

n	δ	Mean Success Rate	Mean Episode Reward	Mean Episode Cost
1	-	0.550	-15.608	21.177
2	0.00	0.579	-15.460	19.089
	0.05	0.570	-15.280	19.756
	0.10	0.579	-15.329	19.588
	0.20	0.570	-15.004	19.510
	0.50	0.570	-15.065	20.797
5	0.00	0.592	-15.125	17.886
	0.05	0.581	-15.273	17.024
	0.10	0.582	-15.382	16.939
	0.20	0.586	-15.174	17.564
	0.50	0.589	-15.014	18.023
10	0.00	0.575	-15.531	16.647
	0.05	0.576	-15.498	16.069
	0.10	0.592	-15.038	16.259
	0.20	0.596	-15.132	16.198
	0.50	0.581	-15.264	16.464

Table 5.5: Analysis of the safety evaluation algorithm with different values for n and δ. The three agents trained for safety evaluation were each evaluated with the different parameters. For $n = 1$ the value of δ makes no difference. The values shown here are the mean of the three agents.

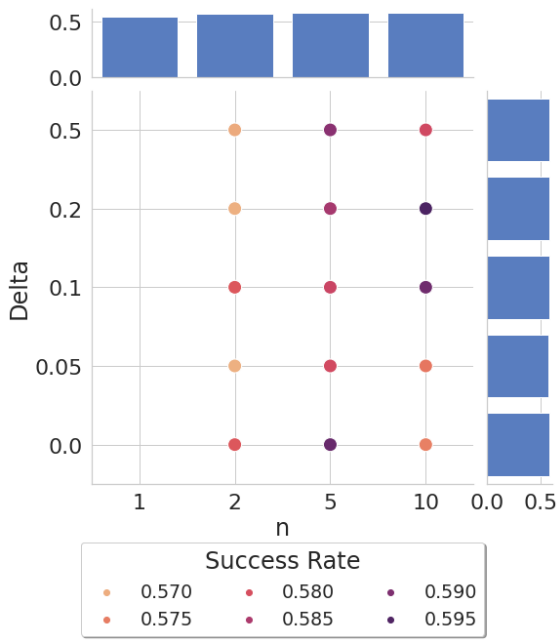
The more actions are drawn the higher the probability to get a good action. However, since the action with the lowest expected cost is selected, the effect on the success rate is only marginal. When looking at the mean episode reward, the difference between the experiments is so little that an interpretation makes no sense.

The value for δ seems to make no difference in all of the experiments. This is explained by a misconception when designing this evaluation. The choices for δ are within the range of the cost of one action, while the safety critic learns the expected cost for a whole episode. Therefore, the expected cost is always higher than δ and the action with the lowest expected cost is selected.

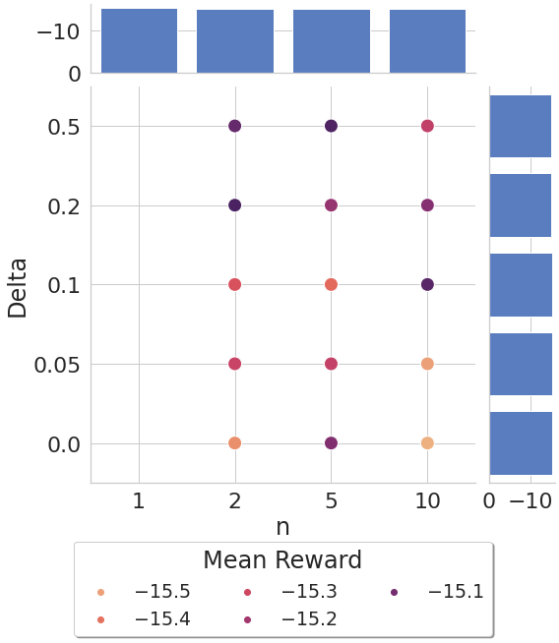
Overall, it can be said that the safety evaluation was able to reduce the cost, with a trade-off between the cost and the execution time. However, the cost is still a lot higher than the cost produced by an agent trained with the safety training algorithm or the reward based approach.

5.3.4 FetchPushBarrier Pixel

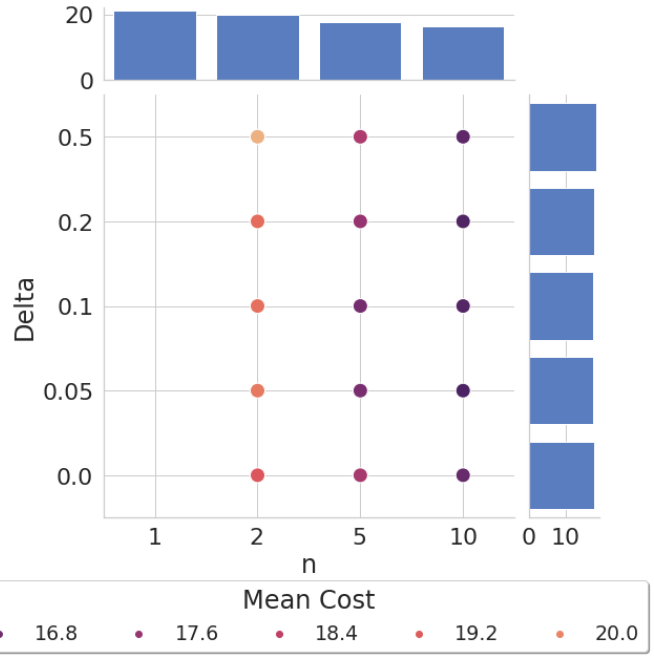
The final experiment was conducted on the FetchPushBarrier environment using image observations. It uses the combination of the preference reward algorithm and a cost representation. Experiments were conducted with the reward based approach and the



(a) Safety evaluation success rate



(b) Safety evaluation reward



(c) Safety evaluation reward

Figure 5.9: Analysis of the safety evaluation algorithm with different values for n and δ. The three agents trained for safety evaluation were each evaluated with the different parameters. For $n = 1$ the value of δ makes no difference. The subplots (a)-(c) show the success rate, mean reward and mean cost for all parameter combinations. The bar plots on the side show the mean values for the specific values for n or δ, averaged over the other parameter.

Algorithm	α	Mean Success Rate	Mean Episode Reward	Mean Episode Cost
Reward Based	0.5	0.003	-45.532	2.425
	1.0	0.003	-52.985	6.809
Safety Training	0.5	0.003	-40.041	3.080
	1.0	0.001	-46.977	2.220

Table 5.6: Mean success rate and reward for all agents trained on the FetchPushBarrier environment using pixels. For each setup only one agent was trained. All agents were evaluated on 1000 episodes.

safety training, as these had the highest success rates in the previous experiments. When using the reward based approach, first the reward and cost are combined and then the preference reward is calculated. Unlike in definition 4.2(1), the environment reward r_{env} is not given by the MDPs reward function, but by the combination of the reward and the cost function $R_{cost}(s, a, s')$. See equation 4.1 for the definition of the function. The combined reward function is then given by:

$$r_{pr} = \alpha * r_{teacher} + (1 - \alpha) * \left(\lambda_r R(s, a, s') - \lambda_c C(s, a, s')\right)$$

Again, the parameters λ_r and λ_c for the reward based approach were both set to one. We trained agents using α values of 0.5 and one. The parameter α defines the importance of the teacher in contrast to the environment reward.

After the training, we evaluated the agents on 1000 episodes. The results are shown in table 5.6. None of the agents was able to reach a meaningful success rate. However, all of the agents where able to solve the task at least once. In contrast to the FetchPush environment, the FetchPushBarrier environment is always initialized in a way that requires the agent to move the object to solve the task. This indicates that the agents did learn some behavior required to solve the task but not quite how to solve it. This theory is supported when looking at videos from the evaluation. Figure 5.10 shows 5 frames from a video recorded during the evaluation of the agent trained with safety training and an α of 0.5. The agent learned to move the object around the barrier and towards the target, but does not move it quite far enough. The episode is therefore not considered a success.

For the sake of completeness, figure 5.11 shows the development of the cost during the training. However, since the goal for the agents is to solve the task while keeping the cost low, it makes little sense to interpret the cost if the task is not solved. An agent that does not move the object and always keeps a distance to the barrier could achieve minimal cost, but would never solve the task. Because there is little to interpret with regards to the result for this experiment, we only trained one agent for each parameter combination.

We see two reasons why we were not able to reliably solve the FetchPushBarrier environment using high dimensional image observations. The first reasons is the difficulty of the environment itself. Even when training on state observations, our agents where only able to reach a mean success rate of 68%. The second reason is connected to the first. The teacher we used is an ensemble of SAC state agents. We trained six agents in total

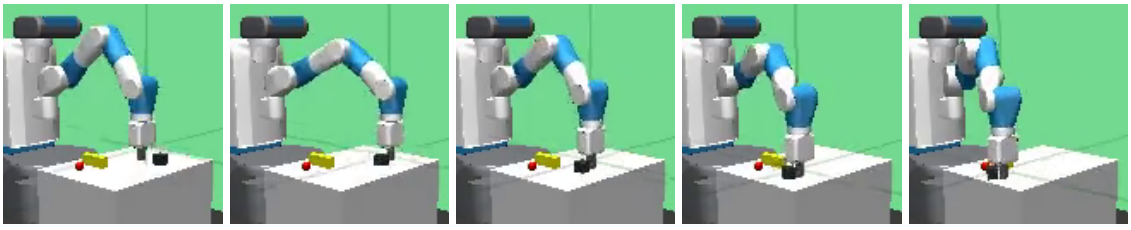


Figure 5.10: Frames from a video recorded on the FetchPushBarrier - Pixel environment. The video was recorded with the agent trained using safety training and an α of 0.5. Between each image, 5 frames where skipped.

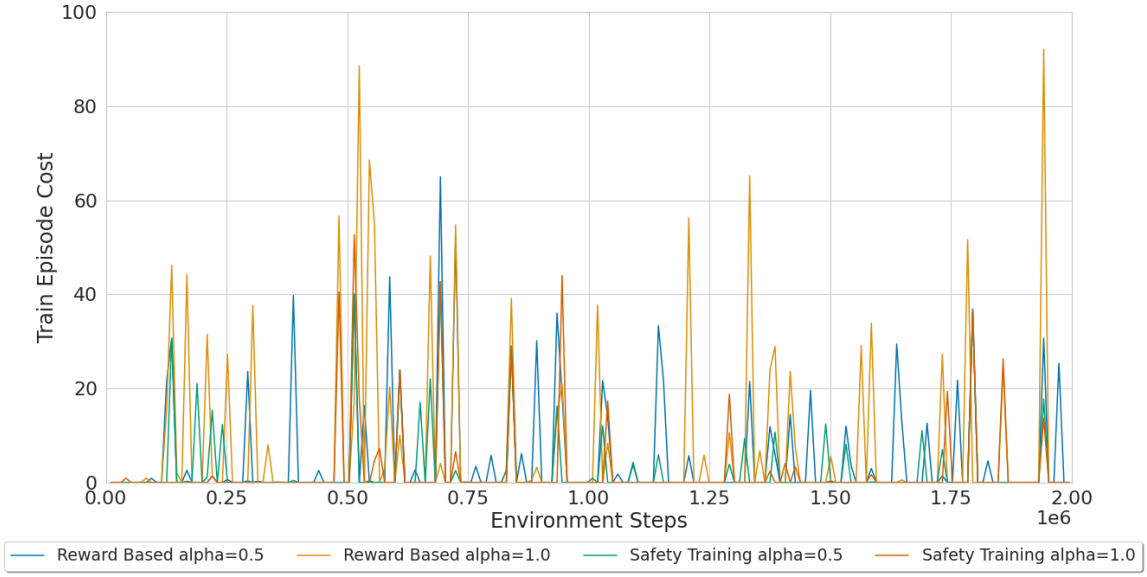


Figure 5.11: The episode cost from the FetchPushBarrier pixel experiments. The FetchPushBarrier environment was trained with the reward based approach as well as the safety training.

using SAC state and selected the three best agents. By doing so, we were able to increase the mean success rate of the teacher to 76.1%. This can be seen in table 5.1. However, this is significantly worse than the teacher for FetchReach and FetchPush. Figure 5.12 shows the success rates in relation to each other. Additionally, it shows the mean confidence of the teachers. To calculate the mean confidence, first the best agent of each ensemble, with regards to the success rate, was selected. This agent was evaluated on 1000 episodes. For each action, the confidence was calculated using all three agents. The mean of the confidence of all actions is shown in the chart. The teacher for the FetchPushBarrier environment also has the lowest mean confidence. This means that the agents of the ensemble propose different actions as the best action. However, this makes it hard for the student to learn the best action. The most drastic example would be, if two of the teacher agents want to pass the barrier on different sides. It is up to future work to show if an improved teacher can lead to better performance on the FetchPushBarrier environment using high dimensional image observations.

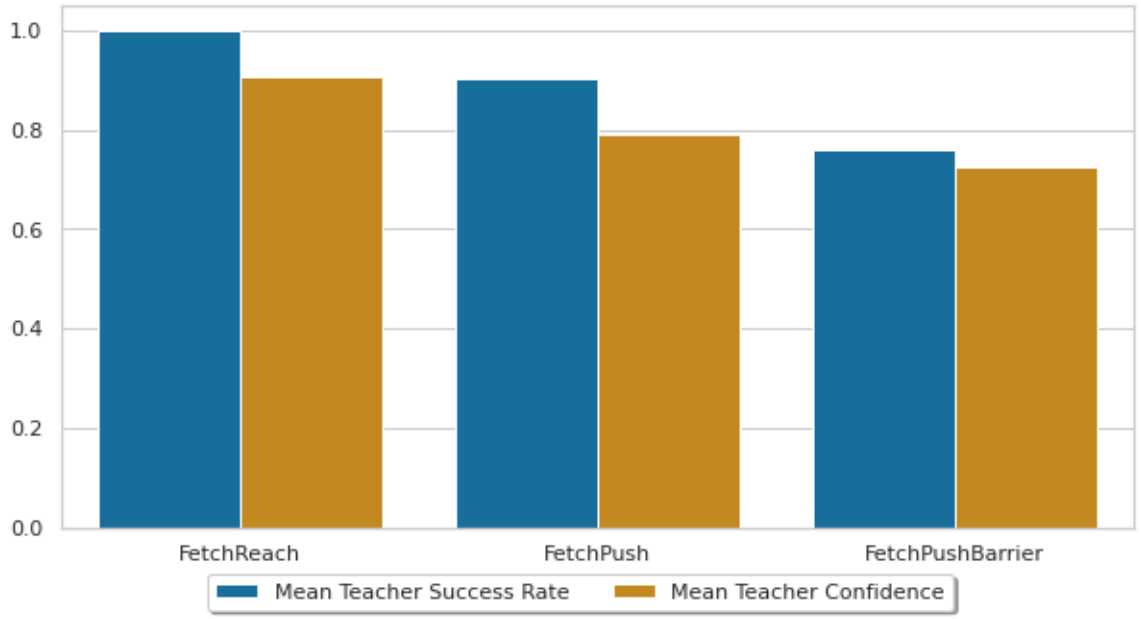


Figure 5.12: Comparison of the mean success rate and confidence of the teachers. For each environment, all three agents that form the ensemble teacher where evaluated on 1000 episodes, to calculate the mean success rate. To calculate the mean confidence, first the best agent of each ensemble, with regards to the success rate, was selected. This agent was evaluated again on 1000 episodes. The environment was initialized with the same seed. For each action, the confidence was calculated using all three agents. The mean of the confidence of all actions is shown here.

6 Conclusion

This work had two goals. The first goal was to introduce a teacher to the reinforcement learning framework. The second was to develop new algorithms to train RL agents that safely interact with their environment.

The teacher was introduced with the preference reward algorithm, a reward shaping technique that combines the reward signal from the environment with a new reward function for the teacher. The teacher was defined as a function that maps a state to an action and a confidence about the quality of this action. The teachers action is then compared to the students action using the mean square error, to form the teacher reward.

For the second goal, we utilized a safety critic. Similar to the reward critic from SAC, the safety critic is trained to estimate the expected cost for the remainder of the episode. Based on the safety critic we proposed to new approaches and compared them to an established reward based approach that deducts the cost from the reward when training the agent. The two safety critic based approaches are safety training and safety evaluation. The safety training algorithm changes the training objective of SAC, to optimize the reward and cost together. In contrast, the safety evaluation algorithm does not change the training. The goal with this algorithm was not to reduce the cost during training, but to learn how to avoid cost during evaluation. To do so, we utilized the stochastic nature of SAC policies to draw multiple actions from the policy and execute the one with lowest cost.

Afterwards we evaluated the three algorithms on two environments form the OpenAI Gym robotics environments as well as a custom environment. The two environment from OpenAI Gym were FetchReach and FetchPush. Based on the FetchPush environment we developed the FetchPushBarrier environment, where the object and the target are always separated by a barrier, the agent has to move the object around.

On the FetchReach environment, we saw that adding a teacher lead to improvements in the success rate when training SAC using pixels and also when adding an autoencoder as an auxiliary loss. When using DrQ to improve the performance of SAC pixel, the agent was able to solve the FetchReach environment without the help of teacher. However, with a teacher we were able to significantly improve the sample efficiency.

For the experiments on the FetchPush environment, we only trained agents with DrQ, since only those were able to reliably solve the FetchReach environment. We saw that an agent trained without the teacher was not able to solve the environment. We then continued to train multiple agents with different values for α, the parameter that marks the importance of the teacher. We saw that only trusting the teacher and not using the environment reward at all did not lead to the best results. With an α value of 0.5 we achieved the best results. We achieved similar results with an adaptive α. A function that reduces the α value when the environment reward increases, indicating that the agent learned something.

On the FetchPushBarrier environment, we first trained agents using the full state information, to evaluate the safety training and safety evaluation algorithms. We saw that agents trained with the safety training algorithm were able to outperform the agents trained with the reward based approach we used to compare our new algorithms to. With the safety, training we saw similar, but slightly better success rates, as well as significantly lower episode costs. The agents trained using the safety evaluation algorithm, however, lacked behind, both in the success rate and the episode costs. While the safety evaluation was able to reduce the episode costs in contrast to not using it, the costs are still much higher than using the reward based approach or the safety training algorithm.

Lastly, we performed experiments on the FetchPushBarrier environment using high dimensional image observations. We were not able to reliably solve the task with any configuration we tried. The configuration varied in the safety algorithm we used, as well as the α value. However, we saw that the agents did learn to move the object and even move it around the barrier. We suspect that this is due to the lower quality of the teacher when compared to the teachers of the FetchReach and FetchPush environment. We leave it up to future work to show if an improved teacher can lead to better performance on the FetchPushBarrier environment using high dimensional image observations.

Overall, we saw significant improvements in the sample efficiency or success rate on the FetchReach and FetchPush environments when using the teacher. We were not able to solve the FetchPushBarrier environment when using pixel observations, even with a teacher. However, it unclear if this due to the difficulty of the environment or the fact that the teacher for the FetchPushBarrier environment is significantly worse than the teachers for the other environments. This leaves room for future work to explore the performance on the FetchPushBarrier environment with an improved teacher. A possible solution could be as simple as improving the performance of SAC state on the FetchPushBarrier environment. This could be done for example by using a hindsight experience replay buffer [2].

In the following we will propose further improvements and experiments that can be explored in future work. Similar to the improved ensemble teacher mentioned above, other teachers could improve the performance. This includes classic control methods like trajectory planners as well as human based teachers. The benefit of these teachers is that they are easier to use in real world scenarios, where it might not be viable to first train multiple agents using the state space. However, these teachers require a new approach to measure the confidence in the actions.

Other improvement we see are in the choice of the parameter α. In this work we either set a fixed value or used a linear function based on the environment reward. The first step could be to try different functions that for example decrease the value faster as the environment reward increases. Also another metric could be used to measure how much the agent has learned already and lower the α value accordingly. Another possibility is to directly optimize the α value, similar to the temperature in SAC that defines the importance of the entropy in contrast to the reward. As described in section 5.3.2, an adaptive α can lead to lower preference rewards even when the agent performs better with regards to the environment reward. With off-policy RL algorithms this is not ideal for training. We see a possible solution to this problem by storing both the environment and the teacher reward in the replay buffer. The preference reward is then calculated

only when it is needed with the current α value. However, we leave the evaluation of this possible solution for future work.

The most drastic improvement we propose is the combination of the preference reward algorithm with model base RL. In model based RL, a dynamics model is learned to predict the next state of the environment before an action is executed. The learned dynamics model could be used to predict the effect of the teachers action and therefore measure the quality. The importance of the teacher can then be set according to the predicted quality.

Finally, it would be interesting to see the impact of the preference reward algorithm on experiments with real robots. As stated above, it is probably required to utilize a different teacher to the ensemble teacher, used in this work, to avoid the need to train multiple RL agents. The ensemble teacher would also require a lot of additional sensors to measure the state space we had available in the simulation.

Bibliography

[1] Joshua Achiam et al. *Constrained Policy Optimization*. May 30, 2017. arXiv: 1705.10528 [cs]. URL: http://arxiv.org/abs/1705.10528 (visited on 07/18/2022).

[2] Marcin Andrychowicz et al. *Hindsight Experience Replay*. Feb. 23, 2018. DOI: 10.48550/arXiv.1707.01495. arXiv: 1707.01495 [cs]. URL: http://arxiv.org/abs/1707.01495 (visited on 11/21/2022).

[3] Homanga Bharadhwaj et al. *Conservative Safety Critics for Exploration*. Apr. 26, 2021. arXiv: 2010.14497 [cs, stat]. URL: http://arxiv.org/abs/2010.14497 (visited on 07/31/2022).

[4] V. S. Borkar. "An Actor-Critic Algorithm for Constrained Markov Decision Processes". In: *Systems & Control Letters* 54.3 (Mar. 1, 2005), pp. 207–213. ISSN: 0167-6911. DOI: 10.1016/j.sysconle.2004.08.007. URL: https://www.sciencedirect.com/science/article/pii/S0167691104001276 (visited on 08/08/2022).

[5] Gabriel de la Cruz et al. "Initial Progress in Transfer for Deep Reinforcement Learning Algorithms". In: July 1, 2016.

[6] Yarin Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. Oct. 4, 2016. arXiv: 1506.02142 [cs, stat]. URL: http://arxiv.org/abs/1506.02142 (visited on 09/22/2022).

[7] Shani Gamrian and Yoav Goldberg. *Transfer Learning for Related Reinforcement Learning Tasks via Image-to-Image Translation*. July 4, 2019. arXiv: 1806.07377 [cs]. URL: http://arxiv.org/abs/1806.07377 (visited on 08/11/2022).

[8] Shangding Gu et al. *A Review of Safe Reinforcement Learning: Methods, Theory and Applications*. May 20, 2022. DOI: 10.48550/arXiv.2205.10330.

[9] Tuomas Haarnoja et al. "Learning to Walk via Deep Reinforcement Learning". June 19, 2019. arXiv: 1812.11103 [cs, stat]. URL: http://arxiv.org/abs/1812.11103 (visited on 03/19/2022).

[10] Tuomas Haarnoja et al. "Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: (), p. 10.

[11] Dmitry Kalashnikov et al. *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*. Nov. 27, 2018. DOI: 10.48550/arXiv.1806.10293. arXiv: 1806.10293 [cs, stat]. URL: http://arxiv.org/abs/1806.10293 (visited on 07/25/2022).

[12] Ilya Kostrikov, Denis Yarats, and Rob Fergus. *Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels*. Mar. 7, 2021. arXiv: 2004.13649 [cs, eess, stat]. URL: http://arxiv.org/abs/2004.13649 (visited on 08/01/2022).

[13] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. ISSN: 0003-4851. JSTOR: 2236703.

[14] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. Dec. 19, 2013. arXiv: 1312.5602 [cs]. URL: http://arxiv.org/abs/1312.5602 (visited on 09/20/2022).

[15] Santiago Paternain et al. *Constrained Reinforcement Learning Has Zero Duality Gap*. Oct. 29, 2019. arXiv: 1910.13393 [cs, math, stat]. URL: http://arxiv.org/abs/1910.13393 (visited on 08/08/2022).

[16] Santiago Paternain et al. *Safe Policies for Reinforcement Learning via Primal-Dual Methods*. Jan. 12, 2022. arXiv: 1911.09101 [cs, eess, math]. URL: http://arxiv.org/abs/1911.09101 (visited on 08/08/2022).

[17] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. *OptLayer - Practical Constrained Optimization for Deep Reinforcement Learning in the Real World*. Feb. 23, 2018. arXiv: 1709.07643 [cs]. URL: http://arxiv.org/abs/1709.07643 (visited on 08/11/2022).

[18] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. *CURL: Contrastive Unsupervised Representations for Reinforcement Learning*. Sept. 21, 2020. arXiv: 2004.04136 [cs, stat]. URL: http://arxiv.org/abs/2004.04136 (visited on 08/01/2022).

[19] Adam Stooke et al. "Decoupling Representation Learning from Reinforcement Learning". In: (), p. 10.

[20] Richard S. Sutton. *Reinforcement Learning: An Introduction*. Ed. by Andrew Barto. Second edition. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2018. xxii, 526 Seiten : Illustrationen, Diagramme ; 24 cm. ISBN: 978-0-262-03924-6.

[21] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. *Reward Constrained Policy Optimization*. Dec. 26, 2018. arXiv: 1805.11074 [cs, stat]. URL: http://arxiv.org/abs/1805.11074 (visited on 07/31/2022).

[22] Akifumi Wachi and Yanan Sui. *Safe Reinforcement Learning in Constrained Markov Decision Processes*. Aug. 14, 2020. arXiv: 2008.06626 [cs]. URL: http://arxiv.org/abs/2008.06626 (visited on 08/11/2022).

[23] Nolan Wagener, Byron Boots, and Ching-An Cheng. *Safe Reinforcement Learning Using Advantage-Based Intervention*. July 19, 2021. arXiv: 2106.09110 [cs, eess]. URL: http://arxiv.org/abs/2106.09110 (visited on 08/11/2022).

[24] *Welcome to Spinning Up in Deep RL! — Spinning Up Documentation*. URL: https://spinningup.openai.com/en/latest/ (visited on 09/26/2022).

[25] Tsung-Yen Yang et al. *Projection-Based Constrained Policy Optimization*. Oct. 7, 2020. arXiv: 2010.03152 [cs]. URL: http://arxiv.org/abs/2010.03152 (visited on 07/18/2022).

[26] Denis Yarats et al. "Improving Sample Efficiency in Model-Free Reinforcement Learning from Images". July 9, 2020. arXiv: 1910.01741 [cs, stat]. URL: http://arxiv.org/abs/1910.01741 (visited on 02/27/2022).

[27] Ming Yu et al. *Convergent Policy Optimization for Safe Reinforcement Learning*. Oct. 26, 2019. arXiv: 1910.12156 [cs, stat]. URL: http://arxiv.org/abs/1910.12156 (visited on 08/08/2022).

[28] Albert Zhan et al. *A Framework for Efficient Robotic Manipulation*. Dec. 14, 2020. arXiv: 2012.07975 [cs]. URL: http://arxiv.org/abs/2012.07975 (visited on 07/04/2022).